ClientAce Help

Table of Contents

Help version 1.102

Contents

Overview

ClientAce provides tools to help developers easily build an OPC client application. ClientAce consists of two main parts: the .NET Application Programming Interface (API) and the DA Junction. Descriptions of the parts are as follows:

- **ClientAce .NET API:** The ClientAce .NET API provides C# and Visual Basic .NET language users with a simple, intuitive, and optimized class library to quickly develop OPC client applications for accessing OPC servers. For more information, refer to **ClientAce .NET API Assembly**.
- **ClientAce DA Junction:** The ClientAce DA Junction is a customized .NET control that enables Visual Basic .NET or C# programmers to develop OPC client applications that can access a variety of OPC servers. No detailed knowledge of OPC Data Access (DA) interfaces is required. The DA Junction will perform the connection handling procedure between the custom client application and the OPC server, and will monitor and reconnect when necessary. For more information, refer to **ClientAceDA_Junction**.

  Note: When building advanced custom OPC client applications that require more control over OPC functionality, the ClientAce .NET API is recommended.

Additional Controls

For more information on other controls that can be used in the Visual Studio Environment, refer to **Additional Controls**.

System and Application Requirements

Software Requirements

Microsoft Windows operating system requirements are the same for both ClientAce and the Microsoft Visual Studio development environment that is used to develop ClientAce applications. If the operating system's requirements for the version of Visual Studio being used does not list the operating system intended for use, then ClientAce is not supported for use on that operating system.

UAC on Windows Vista and Windows 7

To ensure that all components function correctly in the design environment, turn off UAC on the machines being used to develop applications with ClientAce. UAC limits access to folders and files in the design environment, which will affect some objects in the design environment. UAC does not affect these objects in the Runtime environment.

Hardware Requirements

100 MB available disk space is required. For additional hardware requirements, refer to the Microsoft .NET Framework documentation of the version that will be used in the Visual Studio project.

Microsoft Visual Studio Requirements

ClientAce currently supports the following versions of Microsoft Visual Studio:

- Visual Studio 2003 with the .NET 2.0 Framework (ClientAce Version 1.0)
- Visual Studio 2005 with the .NET 2.0 Framework (ClientAce Version 1.0)
- Visual Studio 2008 SP1 with the .NET 3.5 Framework (ClientAce Version 3.5)
- Visual Studio 2010 with the .NET 4.0/4.5 Framework (ClientAce Version 4.0)
- Visual Studio 2012 with the .NET 4.0/4.5 Framework (ClientAce Version 4.0)
- Visual Studio 2013 with the .NET 4.0/4.5 Framework (ClientAce Version 4.0)

For the installation to complete, the following Visual Studio Environment Settings are required:

- C Sharp (C#) must be installed.
- A default development environment must be configured for the current user.

Notes:

1. ASP.NET applications cannot be developed with ClientAce.
2. Only Windows Applications (EXEs) can be signed. If DLLs are created, the application must be signed. The user running the sign process must have permissions to modify the application being signed."

Supported OPC Specifications

ClientAce supports functionality with the following OPC specifications:

- OPC DA 2.0
- OPC DA 2.05A
- OPC DA 3.0
- OPC Unified Architecture (UA)*
- OPC XML-DA

*ClientAce currently supports OPC UA for use with the OPC DA information model.

Note: Other OPC specifications are not supported at this time.

Runtime Requirements

.NET Framework Requirements

When deploying the custom client applications created using ClientAce, the .NET Framework requirements depend on the version of Visual Studio that was used for development. For more information, refer to **Deploying Your Client Application**.

Visual Studio 2010 C++ Runtime Redistributables

Part of the low-level OPC layer for ClientAce is written using C++ in Visual Studio 2010. It has a dependency on the redistributables for that version. It is important that these files be present when deploying custom client application created using ClientAce. The installer for the redistributables may be located in the ClientAce **Install** folder.

OPC Foundation Core Redistributables

OPC DA client/server connectivity requires core OPC components, which are typically with an OPC server or client. When deploying a custom client application created using ClientAce to a PC that has never had an OPC server or client installed, the core components must be installed for it to work. The installer for the OPC Foundation Core Redistributables may be located in the ClientAce **Install** folder.

ClientAce .NET API Assembly

Kepware's ClientAce .NET API provides developers working with the C# and Visual Basic .NET languages with a simple, intuitive, and optimized class library to quickly develop OPC client applications for accessing OPC servers. ClientAce provides the following:

- A simple, intuitive .NET interface that does not require knowledge of the different OPC DA, UA, and XML-DA interfaces.
- An API that covers the different OPC base technologies (such as COM and DCOM). It also manages the connection handling to multiple OPC Servers, including connection establishment, connection monitoring, and reconnection after errors.
- Simple development of OPC Client applications with C# or Visual Basic .NET.
- Conversion of OPC data from different OPC DA, UA, and XML-DA interfaces into .NET data types.
- Fast searching for both local and remote OPC COM servers.
- High performance and optimized client/server communication through kernel functionality (implemented in C++).

Note: For more information, refer to **EndPointIdentifier Class** and **Kepware.ClientAce.OpcDaClient Namespace**.

EndPointIdentifier Class

OPC UA servers require more connection information than traditional OPC servers do, especially for secured connections. The EndPointIdentifer Class specifies secured connection information, such as server certificates, the security policy, and the message security mode used for UA communication. Its values may be entered manually or obtained through the OpcServerEnum.EnumComServer Method when a UA discovery server is used or the OpcServerEnum.getEndpoints Method when receiving connection information directly from the endpoint.

Property	Data Type	Description
ApplicationName	String	The name of the application.
ApplicationUri	String	The URI of the application.
MessageSecurityMode	Byte	The mode of method security. Possible message security mode values are as follows: None = 1 Sign = 2 SignAndEncrypt = 3
ProductUri	String	The product URI.
SecurityPolicyUri	String	The URI of the security policy. Valid policy URI strings used for UA communications are as follows: http://opcfoundation.org/UA/SecurityPolicy#None http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15 http://opcfoundation.org/UA/SecurityPolicy#Basic256
ServerCertificate	Byte[]	The server certificate.

Note: This class is similar to ServerIdentifer. For more information, refer to **ServerIdentifier Class**.

PkiCertificate Class

New Class Constructor With Eleven Overloads

```
New(
        URI As String,
        HostIP As String,
        DNS As String,
        timeValid As Integer,
        commonName As String,
        organization As String,
        organizationUnit As String,
        locality As String,
        state As String,
```

```
        country As String,
        keyStrength as Integer
)
```

Properties

The PkiCertificate Class allows the creation and encapsulation of an X509 certificate, and provides methods for loading and saving certificates through the WindowsCertificateStore. Once constructed, the certificate's properties are Read Only.

Property	Data Type	Description
URI	String	The ApplicationURI used to identify the application. It must be a unique identifier like "urn:<HostName>:<Company>:<ProductName>".
HostIP	String	The IP Address of the host where the application is running. This field will only be used if the host name is not available.
DNS	String	The name of the host where the application is running.
TimeValid	Integer	The certificate's validity time (in seconds).
CommonName	String	The certificate's display name.
Organization	String	The issuer's organization or company.
OrganizationUnit	String	The issuer's organization unit.
Locality	String	The issuer's location.
State	String	The state where the issuer is located.
Country	String	The country code (such as US or DE).
KeyStrength	Integer	The length of the key (such as 1024 bit or 2048 bit).

Example Code

```
[Visual Basic]
' Declare our Pki Certificate
Dim clientCertificate As Kepware.ClientAce.OpcCmn.PkiCertificate
' Create a new Pki Certificate with constructor
clientCertificate = New Kepware.ClientAce.OpcCmn.PkiCertificate( _
"OPCUA Sample Application:Kepware:OpcUaSampleApplication", _
        "127.0.0.1", _
        "", _
        31536000, _
        "CERT_TEST", _
        "Kepware Technologies", _
        "Development", _
        "Portland", _
        "Maine", _
        "United States", _
        1028)
' If a problem occurred we will receive a null certificate
if IsNothing(clientCertificate) Then
MsgBox("A problem occurred when attempting to create a certificate")
End If
```

```
[C#]
// Create a new Pki Certificate with constructor
PkiCertificate clientCert = new Kepware.ClientAce.OpcCmn.PkiCertificate(
"OPCUA Sample Application:Kepware:OpcUaSampleApplication",
        "127.0.0.1",
        "",
        31536000,
        "CERT_TEST",
        "Kepware Technologies",
        "Development",
```

```
            "Portland",
            "Maine",
            "United States",
            1028);
// If a problem occurred we will receive a null certificate
if (clientCert == null)
{
MessageBox.Show("A problem occurred when attempting to create a certificate");
}
```

toDER Method

Method

```
toDER() As/Returns Byte()
```

Properties

This method outputs a DER-encoded byte array that contains the PkiCertificate. This format is used in the EndpointIndentifier and ConnectInfo classes.

Example Code

```
[Visual Basic]
' Declare our Application URI needed to search for the certificate
Dim URI As String = "OPCUA Sample Application:Kepware:OpcUaSampleApplication"
' Declare our ConnectInfo object
Dim connectInfo As Kepware.ClientAce.OpcDaClient.ConnectInfo = Nothing
' Declare our certificate
Dim clientCertificate As Kepware.ClientAce.OpcCmn.PkiCertificate
' Grab our certificate with private key from the Windows Certificate Store
clientCertificate = Kepware.ClientAce.OpcCmn.PkiCertificate.fromWindowsStoreWithPrivateKey
(Kepware.ClientAce.OpcCmn.WinStoreLocation.LocalMachine, _
            "UA Applications", _
            URI)
'Use the toDER method to pass the certificate to the connectInfo object
connectInfo.ClientCertificate = clientCertificate.toDER()
```

```
[C#]
// Declare our Application URI needed to search for the certificate
String URI = "OPCUA Sample Application:Kepware:OpcUaSampleApplication";
// Declare our ConnectInfo object
Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = null;
// Declare our certificate
Kepware.ClientAce.OpcCmn.PkiCertificate clientCertificate = null;
// Grab our certificate with private key from the Windows Certificate Store
clientCertificate =
Kepware.ClientAce.OpcCmn.PkiCertificate.fromWindowsStoreWithPrivateKey
(Kepware.ClientAce.OpcCmn.WinStoreLocation.LocalMachine,
            "UA Applications",
            URI);
// Use the toDER method to pass the certificate to the connectInfo object
connectInfo.ClientCertificate = clientCertificate.toDER();
```

fromDER Method

Method

```
fromDER(
DERdata as byte()
) As/Returns Kepware.ClientAce.OpcCmn.PkiCertificate
```

Properties

This static method creates a PkiCertificate object from a DER-encoded byte array that contains a certificate. If the fromDER Method fails, a certificate that contains properties as null strings will be returned.

Parameter	Use	Data Type	Description
DERdata	Input	Byte[]	A DER-encoded byte array that contains a certificate.

Example Code

```
[Visual Basic]
' Our DER encoded byte array containing our certificate
Dim DERdata() As Byte
' Creating a PKI Certificate from our DER encoded byte array
Dim certificate As Kepware.ClientAce.OpcCmn.PkiCertificate
certificate = Kepware.ClientAce.OpcCmn.PkiCertificate.fromDER(DERdata)
```

```
[C#]
// Our DER encoded byte array containing our certificate
Byte[] DERdata;
// Creating a PKI Certificate from our DER encoded byte array
Kepware.ClientAce.OpcCmn.PkiCertificate certificate;
certificate = Kepware.ClientAce.OpcCmn.PkiCertificate.fromDER(DERdata);
```

toWindowsStore Method

Method

```
toWindowsStore(
        WindowsCertificateStore as Kepware.ClientAce.OpcCmn.WinStoreLocation,
        storeName as String
)
```

Properties

This method exports the PkiCertificate to the WindowsCertificateStore without the Private Key.

Note: When a secure UA connection is needed, the Private Key must be supplied to the ConnectInfo Object.

Parameter	Use	Data Type	Description
WindowsCertificateStore	Input	Kepware.ClientAce.OpcCmn.WinStoreLocation	This specifies the Windows Certificate Store location where the PkiCertificate will be placed.
storeName	Input	String	This specifies the folder name within the Certificate Store where the certificate will be placed. Most UA applications use the store name "UA Applications".

Note: Additional permissions and/or configuration may be necessary when specifying a Windows Store Location besides "Current User" and "Local Computer".

Example Code

```
[Visual Basic]
' Create a certificate
Dim clientCertificate As New Kepware.ClientAce.OpcCmn.PkiCertificate( _
"OPCUA Sample Application:Kepware:OpcUaSampleApplication", _
        "127.0.0.1", _
```

```
            "", _
            31536000, _
            "ClientAce UA Sample Application", _
            "Kepware Technologies", _
            "Development", _
            "Portland", _
            "Maine", _
            "United States", _
            1028)
' Specify a Windows Store Location
Dim winStoreLocation As Kepware.ClientAce.OpcCmn.WinStoreLocation = _
Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser
' Specify a store name. Most UA applications will use the "UA Applications" location.
Dim storeName As String = "UA Applications"
' Export Certificate to the Windows Store
clientCertificate.toWindowsStore(winStoreLocation, storeName)
```

```
[C#]
// Create a certificate
Kepware.ClientAce.OpcCmn.PkiCertificate clientCertificate =
new Kepware.ClientAce.OpcCmn.PkiCertificate(
"OPCUA Sample Application:Kepware:OpcUaSampleApplication",
            "127.0.0.1",
            "",
            31536000,
            "ClientAce UA Sample Application",
            "Kepware Technologies",
            "Development",
            "Portland",
            "Maine",
            "United States",
            1028);
// Specify a Windows Store location
Kepware.ClientAce.OpcCmn.WinStoreLocation
winStoreLocation = Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser;
// Specify a store name. Most UA applications will use the "UA Applications" location.
string storeName = "UA Applications";
// Export Certificate to the Windows Store
clientCertificate.toWindowsStore(winStoreLocation, storeName);
```

toWindowsStoreWithPrivateKey Method

Method

```
toWindowsStoreWithPrivateKey(
            WindowsCertificateStore as Kepware.ClientAce.OpcCmn.WinStoreLocation,
            storeName as String
)
```

Properties

This method exports the PkiCertificate (including the Private Key) to the WindowsCertificateStore.

Parameter	Use	Data Type	Functionality
WindowsCertificateStore	Input	Kepware.ClientAce.OpcCmn.WinStoreLocation	This specifies the Windows Certificate Store location where the PkiCertificate will be placed.

storeName	Input	String	This specifies the folder name within the Certificate Store where the certificate will be placed. Most UA applications use the store name "UA Applications".

Note: Additional permissions and/or configuration may be necessary when specifying a Windows Store Location besides "Current User" and "Local Computer".

Example Code

```
[Visual Basic]
' Create a certificate
Dim clientCertificate As New Kepware.ClientAce.OpcCmn.PkiCertificate( _
"OPCUA Sample Application:Kepware:OpcUaSampleApplication", _
        "127.0.0.1", _
        "", _
        31536000, _
        "ClientAce UA Sample Application", _
        "Kepware Technologies", _
        "Development", _
        "Portland", _
        "Maine", _
        "United States", _
        1028)
' Specify a Windows Store Location
Dim winStoreLocation As Kepware.ClientAce.OpcCmn.WinStoreLocation = _
Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser
' Specify a store name. Most UA applications will use the "UA Applications" location.
Dim storeName As String = "UA Applications"
' Export Certificate to the Windows Store
clientCertificate.toWindowsStoreWithPrivateKey(winStoreLocation, storeName)
```

```
[C#]
// Create a certificate
Kepware.ClientAce.OpcCmn.PkiCertificate clientCertificate =
new Kepware.ClientAce.OpcCmn.PkiCertificate(
"OPCUA Sample Application:Kepware:OpcUaSampleApplication",
        "127.0.0.1",
        "",
        31536000,
        "ClientAce UA Sample Application",
        "Kepware Technologies",
        "Development",
        "Portland",
        "Maine",
        "United States",
        1028);
// Specify a Windows Store location
Kepware.ClientAce.OpcCmn.WinStoreLocation
winStoreLocation = Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser;
// Specify a store name. Most UA applications will use the "UA Applications" location.
string storeName = "UA Applications";
// Export Certificate to the Windows Store
clientCertificate.toWindowsStoreWithPrivateKey (winStoreLocation, storeName);
```

fromWindowsStore Method

Method

```
fromWindowsStore(
        winStoreLocation as Kepware.ClientAce.OpcCmn.WinStoreLocation,
        storeName as String,
        thumbprint as Byte[]
) As/returns Kepware.ClientAce.OpcCmn.PkiCertificate
```

Properties

This method retrieves a certificate from a specific location within the Windows Certificate Store by referencing its thumbprint.

Note: When a secure UA connection is needed, the Private Key must be supplied to the ConnectInfo Object. The fromWindowsStoreWithPrivateKey Method is recommended when there are no other private key storage/retrieval methods being used within the client application. For more information, refer to **fromWindowsStoreWithPrivateKey Method**.

Parameter	Use	Data Type	Description
WindowsCertificateStore	Input	Kepware.ClientAce.OpcCmn.WinStoreLocation	This specifies the Windows Certificate Store location where the PkiCertificate will be found.
storeName	Input	String	This specifies the folder name within the Certificate Store where this certificate will be found. Most UA applications use the store name "UA Applications".
thumbprint	Input	Byte[]	This is an attribute of an X509 certificate that can be copied directly from the Certificate Properties within the Windows MMC Certificate snap-in.

Example Code

```
[Visual Basic]
Private Function RetrieveCertificateFromStore(ByRef thumbprint() As Byte)
' Declare a certificate for our return value
Dim certificate As Kepware.ClientAce.OpcCmn.PkiCertificate
' Specify a Windows Store Location
Dim winStoreLocation As Kepware.ClientAce.OpcCmn.WinStoreLocation = _
Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser
' Specify a store name. Most UA applications will use the "UA Applications" location.
Dim storeName As String = "UA Applications"
Try
' Retrieve the certificate form the windows store
certificate = Kepware.ClientAce.OpcCmn.PkiCertificate.fromWindowsStore( _
        winStoreLocation, _
        storeName, _
```

```
        thumbprint)
' Return the certificate from the function
Return certificate
Catch ex As Exception
MsgBox("A handled exception was caught when attempting to retrieve the certificate Form the
WindowsFormsSection store: " _
        & ex.ToString())
' If an exception occurred, return nothing
Return Nothing
End Try
End Function
```

```
[C#]
private PkiCertificate retrieveCertificateFromStore(byte[] thumbprint)
{
// Declare a certificate for our return value
PkiCertificate certificate;
// Specify a Windows Store location
Kepware.ClientAce.OpcCmn.WinStoreLocation
winStoreLocation = Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser;
// Specify a store name. Most UA applications will use the "UA Applications" location.
string storeName = "UA Applications";
try
{
// Retrieve the certificate form the windows store
certificate = PkiCertificate.fromWindowsStore(winStoreLocation,
storeName,
thumbprint);
// Return the certificate
return certificate;
}
catch (Exception ex)
{
MessageBox.Show("A handled exception was caught when attempting to retrieve the certificate Form
the WindowsFormsSection store: "
        + ex.ToString());
// If an exception occurred, return null
return null;
}
}
```

fromWindowsStoreWithPrivateKey Method

Method One

```
fromWindowsStoreWithPrivateKey(
        winStoreLocation as Kepware.ClientAce.OpcCmn.WinStoreLocation,
        storeName as String,
        thumbprint as Byte[]
) As/returns Kepware.ClientAce.OpcCmn.PkiCertificate
```

Method Two

```
fromWindowsStoreWithPrivateKey(
        winStoreLocation as Kepware.ClientAce.OpcCmn.WinStoreLocation,
        storeName as String,
        applicationURI as String
) As/returns Kepware.ClientAce.OpcCmn.PkiCertificate
```

Properties

This method retrieves a certificate with the private key from a specific location within the Windows Certificate Store by referencing the certificate's thumbprint or application URI.

Parameter	Use	Data Type	Description
WindowsCertificateStore	Input	Kepware.ClientAce.OpcCmn.WinStoreLocation	This specifies the Windows Certificate Store location where the PkiCertificate will be found.
storeName	Input	String	This specifies the folder name within the Certificate Store where the certificate will be found. Most UA applications use the store name "UA Applications".
thumbprint	Input-1	Byte[]	This is an attribute of an X509 certificate that can be copied directly from the Certificate Properties within the Windows MMC Certificate snap-in.
applicationURI	Input-2	String	If desired, a string that contains the application URI can be used instead of the thumbprint to retrieve the certificate from the Windows store.

Example Code

```
[Visual Basic]
Private Function RetrieveCertificateFromStore(ByRef thumbprint() As Byte)
' Declare a certificate for our return value
Dim certificate As Kepware.ClientAce.OpcCmn.PkiCertificate
' Specify a Windows Store Location
Dim winStoreLocation As Kepware.ClientAce.OpcCmn.WinStoreLocation = _
Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser
' Specify a store name. Most UA applications will use the "UA Applications" location.
Dim storeName As String = "UA Applications"
Try
' Retrieve the certificate form the windows store
certificate = Kepware.ClientAce.OpcCmn.PkiCertificate.fromWindowsStoreWithPrivateKey ( _
        winStoreLocation, _
        storeName, _
        thumbprint)
' Return the certificate from the function
Return certificate
Catch ex As Exception
MsgBox("A handled exception was caught when attempting to retrieve the certificate Form the
```

```
WindowsFormsSection store: " _
        & ex.ToString())
' If an exception occurred, return nothing
Return Nothing
End Try
End Function
```

```
[C#]
private PkiCertificate retrieveCertificateFromStore(byte[] thumbprint)
{
// Declare a certificate for our return value
PkiCertificate certificate;
// Specify a Windows Store location
Kepware.ClientAce.OpcCmn.WinStoreLocation
winStoreLocation = Kepware.ClientAce.OpcCmn.WinStoreLocation.CurrentUser;
// Specify a store name. Most UA applications will use the "UA Applications" location.
string storeName = "UA Applications";
try
{
// Retrieve the certificate form the windows store
certificate = PkiCertificate.fromWindowsStoreWithPrivateKey(winStoreLocation,
storeName,
thumbprint);
// Return the certificate
return certificate;
}
catch (Exception ex)
{
MessageBox.Show("A handled exception was caught when attempting to retrieve the certificate Form
the WindowsFormsSection store: "
        + ex.ToString());
// If an exception occurred, return null
return null;
}
}
```

ServerIdentifier Class

The ServerIdentifier class is used to instance the server identifiers that are used by the DAServerObject for OPC connections.

New Class Constructor With No Overloads

```
New( )
```

New Class Constructor With Two Overloads

```
New(
        url As String,
        endpoint As Kepware.ClientAce.OpcCmn.EndpointIdentifier
)
```

New Class Constructor With Four Overloads

```
New(
        nodeName As String,
        progId As String,
        clsid As String,
        category As Kepware.ClientAce.OpcCmn.ServerCategory
)
```

Properties

ServerIdentifier objects are returned by the EnumComServers Method and contain information that describe the OPC servers installed on the specified machine.

Property	Data Type	Description
Category	ServerCategory	Server category.*
CLSID	String	The Class ID of the OPC server.
Endpoint	EndpointIdenifier	The Kepware.ClientAce.OpcCmn Endpoint Identifier.**
HostName	String	The name or IP address of the OPC server's host machine (such as localhost, PCTest, 192.168.0.120, and so forth). If this parameter is left unassigned, the local host is assumed.
ProgID	String	The Program ID of the OPC server.
URL	String	The URL of the server, formatted for use in **Connect Method**.

*For more information, refer to **ServerCategory Enumeration**.
For more information, refer to **EndpointIdentifier Class.

ServerCategory Enumeration

The ServerCategory enumerator is used to specify the type of OPC server.

Name	Value	Description
OPCAE	2	Server supports OPC AE 1.10 (Alarms and Events).
OPCDA	0	Server supports OPC DA 2.0, 2.05A, and 3.0 (Data Access).
OPCDX	1	Server supports OPC DX 1.00 (Data Exchange).
OPCHDA	3	Server supports OPC HDA 1.10 (Historical Data Access).
OPCUA	5	Server supports OPC UA 1.01 (Unified Architecture).
OPCXMLDA	4	Server supports OPC XMLDA 1.01 (XML Data Access).

Note: Because OPC XML-DA servers are not registered like COM OPC servers, they cannot be found using the OpcServerEnum object. To connect to an OPC XML-DA server, the URL must be known.

WinStoreLocation Enumeration

The WinStoreLocation enumerator is an enumerated object that specifies WinStoreLocation values.

Name	Value	Description
CurrentService	262144	The object's current service.
CurrentUser	65536	The object's current user.
LocalMachine	131072	The object's local machine.
Services	327680	The object's services.
Users	393216	The object's users.

OpcServerEnum Object

The Kepware.ClientAce.OpcCmn.OpcServerEnum object enumerates the OPC servers installed on a given machine. It also determines the CLSID from an OPC server's ProgID.

Creating OpcServerEnum Object

Before using the OpcServerEnum Class, an instance of the class must be created.

```
[Visual Basic]
Dim opcServerEnum As New Kepware.ClientAce.OpcCmn.OpcServerEnum
```

```
[C#]
OpcServerEnum opcServerEnum = new
Kepware.ClientAce.OpcCmn.OpcServerEnum ();
```

ClsidFromProgID Method

Method

```
ClsidFromProgId (
        ByVal nodeName As String,
        ByVal progID As String,
        ByRef clsid As String
)
```

Properties

The ClsidFromProgID Method is used to obtain an OPC server's Class ID from its Program ID. The server's host machine must be accessible from the client.

Note: This function is not needed to connect to UA servers.

Parameter	Use	Functionality
nodeName	Input	The name or IP address of the OPC Server's host machine (such as localhost, PCTest, 192.168.0.120, and so forth). If this parameter is left unassigned, the local host is assumed.
progID	Input	The Program ID of the OPC server.
clsid	Output	The returned Class ID of the OPC server.

Example Code

```
[Visual Basic]
' Declare variables
Dim opcServerEnum As Kepware.ClientAce.OpcCmn.OpcServerEnum = Nothing
Dim nodeName As String = "localhost"
Dim progId As String = "KEPware.KEPServerEx.V5"
Dim clsid As String = Nothing
Try
' Call ClsidFromProgId API method
opcServerEnum.ClsidFromProgId(nodeName, progId, clsid)
' Handle result
Console.WriteLine("CLSID: " & clsid)
Catch ex As Exception
Console.WriteLine("ClsidFromProgID exception. Reason: " & _
ex.Message)
End Try
```

```
[C#]
// Declare variables
OpcServerEnum opcEnum = new OpcServerEnum();
string nodeName = "localhost";
string progId = "KEPware.KEPServerEx.V5";
string clsid;
try
{
// Call ClsidFromProgId API method
opcEnum.ClsidFromProgId(nodeName, progId, out clsid);
// Handle result
Console.WriteLine("CLSID: {0}", clsid);
}
catch (Exception ex)
{
Console.WriteLine("ClsidFromProgId exception. Reason: {0}", ex);
}
```

EnumComServer Method

Method

```
EnumComServer (
        ByVal nodeName As String,
        ByVal returnAllServers As Boolean,
        ByVal serverCategories() As Kepware.ClientAce.OpcCmn.ServerCategory,
        ByRef servers() As Kepware.ClientAce.OpcCmn.ServerIdentifier
)
```

Properties

The EnumComServer Method is used to determine the OPC servers that are accessible to a ClientAce application. These servers can exist on the same computer as the client application, or on any machine accessible on the network. The results can be filtered according to OPC server category. For more information, refer to **ServerCategory Enumeration**.

Parameter	Use	Functionality
nodeName	Input	The name or IP address of the OPC server's host machine (such as localhost, PCTest, 192.168.0.120, and so forth). If this parameter is left unassigned, the local host is assumed.
returnAllServers	Input	This flag decides whether to return all OPC Servers found on a particular machine. If this parameter is set to true, the array serverCategories will be ignored.
serverCategories	Input	This parameter specifies the types of OPC servers that should be returned.*
servers	Output	This parameter specifies the OPC servers that should be returned.

***See Also:** **ServerCategory Enumeration**.

Example Code

These examples browse for all OPCDA servers installed on localhost.

```
[ Visual Basic]
' Declare parameters
Dim opcEnum As Kepware.ClientAce.OpcCmn.OpcServerEnum = Nothing
Dim nodeName As String = "localhost"
Dim returnAllServers As Boolean = False
Dim serverCatagories(0) As Kepware.ClientAce.OpcCmn.ServerCategory
serverCatagories(0) = New Kepware.ClientAce.OpcCmn.ServerCategory
serverCatagories(0) = Kepware.ClientAce.OpcCmn.ServerCategory.OPCDA
Dim servers() As Kepware.ClientAce.OpcCmn.ServerIdentifier = Nothing
Try
' Call EnumComServer API method
opcEnum.EnumComServer( _
nodeName, _
returnAllServers, _
serverCatagories, _
servers)
' Handle results
Dim server As Kepware.ClientAce.OpcCmn.ServerIdentifier
For Each server In servers
Dim progID As String = server.ProgID
Dim url As String = server.Url
Console.WriteLine("ProgID: " & progID & " url: " & url)
Next
Catch ex As Exception
Console.WriteLine("Handled EnumComServer exception. Reason: " _
& ex.Message)
End Try
```

```
[C#]
// Declare parameters
string nodeName = "localhost";
bool returnAllServers = false;
OpcServerEnum opcEnum = new OpcServerEnum();
ServerCategory[] serverCategories = new ServerCategory[1];
serverCategories[0] = new ServerCategory();
serverCategories[0] = ServerCategory.OPCDA;
ServerIdentifier[] servers;
try
{
// Call EnumComServer API method
opcEnum.EnumComServer(nodeName, returnAllServers, serverCategories, out servers);
// Handle results
foreach (ServerIdentifier server in servers)
{
string progID = server.ProgID;
string url = server.Url;
Console.WriteLine("ProgID: {0} url: {1}", progID, url);
}
}
catch (Exception ex)
{
Console.WriteLine("EnumComServer exception. Reason: {0}", ex);
}
```

getCertificateForEndpoint Method

Method

```
getCertificateForEndpoint( _
        endpointUrl As String, _
        securityPolicyUri As String, _
        messageSecurityMode As Byte, _
        ByRef serverCertificate() As Byte _
)
```

Properties

The getCertificateForEndpoint Method retrieves the certificates for a specified OPC UA Server EndPoint.

Parameter	Use	Data Type	Description
endpointUrl	Input	String	The URL of the Endpoint.
securityPolicyUri	Input	String	The URI of the security policy. Valid policy URI strings used for UA communications are as follows: http://opcfoundation.org/UA/SecurityPolicy#None http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15 http://opcfoundation.org/UA/SecurityPolicy#Basic256
messageSecurityMode	Input	Byte	The mode of message security. Possible message security mode values are as follows: None = 1 Sign = 2 SignAndEncrypt = 3
serverCertificate	Output	Byte	The ByRef server certificate.

Example Code

```
[VB]
```

```
' Create a new OPC Server Enum object to call the getCertificateForEndpoint method
Dim opcEnum As Kepware.ClientAce.OpcCmn.OpcServerEnum = _
New Kepware.ClientAce.OpcCmn.OpcServerEnum
' Specify a server endpoint
Dim endpointURL As String = "opc.tcp://localhost:49320"
' Specify a security policy URI
Dim securityPolicyURI As String = "http://opcfoundation.org/UA/SecurityPolicy#None"
' Specify a message Security Mode
Dim messageSecurityMode As Byte = 1
' Create a byte array which will contain our returned certificate
Dim serverCertificate() As Byte = Nothing
Try
' Make the method call
opcEnum.getCertificateForEndpoint(endpointURL, securityPolicyURI, messageSecurityMode,
serverCertificate)
Catch ex As Exception
MsgBox("An error occurred when attempting to retrieve the certificate from the following endpoint: " _
& endpointURL _
& ex.ToString())
End Try
```

```
[C#]
// Create a new OPC Server Enum object to call the getCertificateForEndpoint method
Kepware.ClientAce.OpcCmn.OpcServerEnum opcEnum =
new Kepware.ClientAce.OpcCmn.OpcServerEnum();
// Specify a server endpoint
string endpointURL = "opc.tcp://localhost:49320";
// specify a security policy URI
string securityPolicyURI = "http://opcfoundation.org/UA/SecurityPolicy#None";
// Specify a message security mode
byte messageSecurityMode = 1;
// Create a byte array which will contain our returned certificate
byte[] serverCertificate;
try
{
// Make the method call
opcEnum.getCertificateForEndpoint(endpointURL,
securityPolicyURI,
messageSecurityMode,
out serverCertificate);
}
catch(Exception ex)
{
MessageBox.Show("An error occurred when attempting to retrieve the certificate from the following
endpoint: "
+ endpointURL
+ ex.ToString());
}
```

getEndpoints Method

Method

```
getEndpoints(
discoveryURL as string,
endpoints() as Kepware.ClientAce.OpcCmn.EndpointIdentifier
)
```

Properties

The getEndpoints Method is used to retrieve connection information from an OPC UA server when an OPC UA Discovery Server is not installed on the remote machine. Because port numbers can vary widely, searching for OPC UA endpoints may be very communication-intensive. As such, it should not be done using standard server enumeration methods like EnumComServer(). If an OPC Discovery Server is installed and running on the remote machine, the EnumComServer Method will return any OPC servers residing on the remote machine, including all OPC UA endpoints. Otherwise, the getEndpoints Method will allow the client to retrieve UA-specific connection information.

Note: When using the getEndpoints Method, the Endpoint URL must contain the OPC UA server's full address, including the port number. For example, "opc.tcp://[IP Address]:[Port]" and "opc.tcp://localhost:49320".

Parameter	Use	Functionality
discoveryURL	Input	This specifies the OPC UA server's address.
endpoints()	Input/Output	This array of EndpointIdentifiers returns information from any UA server.

Example Code

The examples below show how to retrieve pertinent connection information for a remote OPC UA server.

```
[Visual Basic]
'Declare our serverEnum object instance
Dim opcServerEnum As New Kepware.ClientAce.OpcCmn.OpcServerEnum()
'Create an array of endpoint identifiers for our return values
Dim endpoints(0) As Kepware.ClientAce.OpcCmn.EndpointIdentifier
endpoints(0) = New Kepware.ClientAce.OpcCmn.EndpointIdentifier()
'Specify a valid OPC UA endpoint URL
Dim discoveryURL As String = "opc.tcp://localhost:49320"
Try
'Make the method call
opcServerEnum.getEndpoints(discoveryURL, endpoints)
Catch ex As Exception
MsgBox("An error occurred when attempting to retrieve endpoint information" _
& ex.ToString())
End Try
```

```
[C#]
//Declare our serverEnum object instance
Kepware.ClientAce.OpcCmn.OpcServerEnum opcServerEnum =
new Kepware.ClientAce.OpcCmn.OpcServerEnum();
//Create an array of endpoint identifiers for our return values
Kepware.ClientAce.OpcCmn.EndpointIdentifier[] endpoints =
new Kepware.ClientAce.OpcCmn.EndpointIdentifier[0];
//Specify a valid OPC UA endpoint URL
string discoveryURL = "opc.tcp://localhost:49320";
try
{
//Make the method call
opcServerEnum.getEndpoints(discoveryURL, out endpoints);
}
catch (Exception ex)
{
```

```
System.Windows.Forms.MessageBox.Show(
"An error occurred when attempting to retrieve endpoint information" +
ex.ToString());
}
```

Kepware.ClientAce.OpcDaClient Namespace

The DaServerMgt Object provides the following features in the Kepware.ClientAce.OpcDaClient namespace:

- **OPC Server Connection:** The Connect Method is used to connect to the OPC Server; the Disconnect Method is used to release the connection. Because the connection is monitored by ClientAce, the client will be notified of any changes in connection status through ServerStateChanged Events.
- **Data Change Notification:** To avoid cyclic reading, ClientAce API provides tools that notify the client of changes in values. Items can be registered for monitoring through the Subscribe Method, subscriptions can be canceled through the SubscriptionCancel Method, and notifications of changed values are made by the DataChanged Event.

  Note: Items can be added or removed from a subscription at any time through the SubscriptionAddItems and SubscriptionRemoveItems Methods. Subscription properties can also be changed at any time through the SubscriptionModify Method.
- **OPC Data Access Items Read/Write:** The OPC items' values can be changed using the asynchronous WriteAsync and synchronous Write Methods. When subscription is not appropriate, the values can be obtained through the asynchronous ReadAsync and synchronous Read Methods.
- **Information on the Address Space:** The address space Browse Method can be used to search for OPC items. The GetProperties Method can be used to obtain the properties of OPC items.

For more information, select a link from the list below.

BrowseElement Class

The BrowseElement Class contains all the information that was obtained by using the Browse method.

Property	Data Type	Description
HasChildren	Boolean	This will be true if the element has child elements in the address space; otherwise, it will be false.
IsItem	Boolean	This will be true if the element is an OPC Data Access item; otherwise, it will be false.
ItemName	String	The item name of the element.
ItemPath	String	The item path of the element.
ItemProperties	ItemProperties	The properties of the element that were available through the Browse Method.
Name	String	The name of the returned element. This name is generally used

		for displaying the address space in a tree or other structured format.

ConnectInfo Class

Constructor

```
ConnectInfo(
        localID As String,
        retryInitialConnection As Boolean,
        retryAfterConnectionError As Boolean,
        keepAliveTime As Integer
)
```

Properties

A ConnectInfo object is used to pass connection related options to the API. This information determines how the API will monitor and maintain connections, and will also provide language-dependent strings.

Note: For more information, refer to **WinStoreLocation Enumeration**.

Property	Data Type	Description
BrowserInterface	Integer	0 = Default. When selected, the default browse functionality is used. For a COM OPC Data Access server that supports DA2 and DA3 browsing, the DA3 browse interface is used. 1 = ComDA2. When selected, the COM OPC Data Access 2 browse interface is used if available. 2 = ComDA3. When selected, the COM OPC Data Access 3 browse interface is used if available.
CertificateStoreLocation	WinStoreLocation	The location of the certificate store.
CertificateStoreName	String	The name of the certificate store.
ClientCertificate	Byte[]	The client certificate.
ClientName	String	The user-defined name of the client.
ClientPrivacyKey	Byte[]	The client's privacy key.
DefaultNamespaceUri	String	This specifies a default Namespace URI for OPC UA Communications and gives users the ability to eliminate the "ns=[namespace];s=" prefix used to address OPC Items within the UA address space.*
DisableCacheReadForActiveRWGroup	Boolean	When ForceActiveRWGroup and DisableCacheReadForActiveRWGroup are set, the read is from CACHE if MaxAge is greater than the UpdateRate of the ReadWrite group. Default is False.
DoAdviseActiveRWGroup	Boolean	This property is set to advise the Read/Write Group of changes. This allows the Asynchronous Reads and Writes to complete when the ForceActiveRWGroup flag is set.
DoServerCertificateVerify	Boolean	If this flag is set, the API tries to validate the server certificate when connecting with security. If the application handles the certificate management, this flag can be set to False to disable the check.
ForceActiveRWGroup	Boolean	Add the Read/Write group as Active and actively poll the added items to update the cache. Setting this property forces OPC 2.0 reads from CACHE, ignoring the MaxAge setting. Default is False.
KeepAliveTime	Integer	During Runtime, the API continuously checks the availability of the connection to the server. KeepAliveTime represents the time interval (in milliseconds) at which this availability check takes place. The default value is 10,000 ms. The API begins

		reconnection attempts at an interval of two times the KeepAliveTime and incremented by 1 KeepAliveTime up to 10 times KeepAliveTime if the server is not available for a longer time period. The reconnect interval after a shutdown event from the OPC server is one minute. For example, if KeepAliveTime = 10,000 milliseconds, the first reconnect attempt is 20 seconds after check-fail; the second reconnect attempt is 30 seconds after the first; the third reconnect attempt is 40 seconds after the second, and so on up to 100 seconds. From that point on, retries continue every 100 seconds.
LocalID	String	Using LocalID allows a country abbreviation (en-us, en, and so forth) to be passed to the server. When the LocalID is set, the language-dependent return values are returned in the selected language (if supported by the OPC server). If the value cannot be found, the default value is passed to the server.
MessageSecurityMode	Byte	The mode of message security. 1 = None 2 = Sign 3 = SignAndEncrypt
RetryAfterConnectionError	Boolean	If this flag is set, the API attempts to reconnect after a connection loss until the reconnect succeeds. If the connection can be re-established, all handles that were created before the connection loss are valid again. Event handler methods do not have to be re-registered.
RetryInitialConnection	Boolean	If this flag is set to true, the API tries to connect to the server even when the first connect did not succeed.
RWGroupUpdateRate	UInteger	Sets the update rate for the Read/Write group. Default is 2000 milliseconds.
SecurityPolicyURI	Sting	The URI of the security policy. http://opcfoundation.org/UA/SecurityPolicy#None http://opcfoundation.org/UA/SecurityPolicy#Basic128-Rsa15 http://opcfoundation.org/UA/SecurityPolicy#Basic256
ServerCertificate	Byte[]	The server's certificate.
UACallTimeout	Integer	The timeout used for each UA Service call. This is the amount of time the UA interface waits for a response from the server. Default is 10000 milliseconds.
UserIdentity	UserTokenType	This defines the type of user authentication for the connection.**

*The DefaultNamespaceUri for KEPServerEX version 5 is "KEPServerEX". The Namespace URIs can be found within the Server Namespace Array when browsing the server's address space.
For more information, refer to **UserTokenType Enumeration.

Note: Changes in the connection status should be monitored using a ServerStateChanged event handler. Connect is the only method in the DaServerMgt namespace that can be called prior to establishing a connection. This can be tested at any time with the **IsConnected Property**.

DaServerMgt Class

The DaServerMgt Class allows access to an OPC Data Access Server. For a more information on the ClientAce API and its methods, refer to **DaServerMgt Object**.

ItemIdentifier Class

The ItemIdentifier Class is a required parameter of the following methods:

- GetProperties
- Read

- ReadAsync
- Subscribe
- SubscriptionAddItems
- SubscriptionRemoveItems
- Write
- WriteAsync

ItemIdentifier objects are used to identify OPC items within a server. These objects are passed by reference (in/out) in all method calls so that ClientAce may update the properties described below.

Property	Data Type	Description
ClientHandle	Object	ClientAce will reference items in DataChanged, ReadCompleted, and WriteCompleted events by their ClientHandle. A handle can be assigned to access the data storage object for the item. This storage object could be a TextBox control on the GUI or an instance of a custom class defined in the application.
DataType	System.Type	When an ItemItentifier object is first used, the property may specify the data type as which the item value will be received. If the server cannot provide the requested type for this item, ClientAce will indicate this through the ResultID and reset this property to the item's Native or canonical (default) data type. If this property is left unspecified, ClientAce will reset it using the item's canonical (default) data type.
ItemName	String	This property contains the name (ItemID) of an OPC Data Access item.
ItemPath	String	Reserved for future use.
ResultID	ResultID	Whenever an item-specific error occurs during and OPC call (such as unknown ItemName, trying to write to a Read Only item, unsupported data type, and so forth), the error code provided by the server will be placed in the ResultID object for the associated ItemIdentifier. ClientAce will provide additional descriptive information for the error. If a ClientAce API call returns a **ReturnCode** indicating an error, the ResultID of all ItemIdentifiers passed to the method should be examined to see which items failed and why.
ServerHandle	Integer	The API will set this value when the ItemIdentifier is first used. The API can use the ServerHandle to optimize future calls to the OPC server.

ItemResultCallback Class

The ItemResultCallback Class is used in the WriteCompleted event.

Property	Data Type	Description
ClientHandle	Object	This is the client handle of the item specified in the call to WriteAsync. The client uses this handle to access the appropriate storage object for the received data.
ResultID	ResultID*	The class ResultID provides the error code (int), the name (string) and a language-dependent description (string) for the item represented by the ClientHandle. Thus, certain activities can be programmed to react on occurring errors. It is also possible to display the error on the user interface (message box).

*For more information, refer to **ResultID Class**.

ItemProperty Class

ItemProperty objects are used to describe a single property of an OPC item.

Property	Data Type	Description
DataType	System.Type	The data type of the property value.
Description	String	The description of the property. This information can be used when displaying the property in a graphical user interface (such as in a

		Grid Control or a ToolTip).
ItemName	String	The item name of the property (if the OPC server allows properties to be read from and written to an item).
ItemPath	String	The item path of the property (if the OPC server allows properties to be read from and written to an item).
PropertyID	Integer	The identification number of the property.
ResultID	ResultID*	If an error occurred while obtaining the properties, the dedicated error code will be returned within this object.
Value	Object	The value of the property.

*For more information, refer to **ResultID Class**.

ItemValue Class

The ItemValue Class contains an OPC item's value, quality, and timestamp. It is used in the following methods:

- **Read:** This method takes an array of ItemValue objects as an output parameter. The API will allocate and fill the array with the requested item values during the read.
- **Write:** This method takes an array of ItemValue objects as an input parameter. This array must be filled with the values that will be written to the items specified in the corresponding array of ItemIdentifier objects.
- **WriteAsync:** This method takes an array of ItemValue objects as an input parameter. This array must be filled with the values that will be written to the items specified in the corresponding array of ItemIdentifier objects.

Property	Data Type	Description
Quality	QualityID*	The OPC quality of the associated Value. The class QualityID provides the quality code (int), the name (string), and the description (string). It is Read Only, and will be set by the API during reads.
Timestamp	Date	The timestamp of the associated Value. It is Read Only, and will be set by the API during reads.
Value	Object	The value of the item. As an object, it can contain any data type. The value is generally the same type as requested by the corresponding ItemIdentifier. If no type was specified, the value will be provided in its canonical form.

*For more information, refer to **QualityID Class**.

ItemValueCallback Class

ItemValueCallback is derived from the ItemValue Class and is used in DataChanged and ReadCompleted events.

Property	Data Type	Description
ClientHandle	Object	The client handle of the item specified in the call to Subscribe or ReadAsync. The client uses this handle to access the appropriate storage object for the received data.
Quality	QualityID*	The quality associated with the value when it was acquired from the data source. The class QualityID provides the quality code (int), the name (string), and the description (string). It is Read Only, and will be set by the API during reads.
ResultID	ResultID**	The class ResultID provides the error code (int), the name (string), and a language-dependent description (string) for the item represented by the ClientHandle. Certain activity can be programmed to react on eventually occurring errors. It is also possible to display the error on the user interface (message box).
Timestamp	Date	The timestamp of the associated Value. It is Read Only, and will be set by the API during reads.
Value	Object	The value of the item. As an object, it can contain any data type. The value is generally the same type as requested by the corresponding ItemIdentifier. If no type was specified, the value will be provided in its canonical form.

*For more information, refer to **QualityID Class**.
For more information, refer to **ResultID Class.

Note: Quality, Timestamp, and Value are shared from the base class.

QualityID Class

A QualityID object is used to describe the OPC quality of an item's value.

Property	Data Type	Description
Description	String	The description of the quality code. The language depends on the locale.
FullCode	Integer	The full code sent by the server.
IsGood	Boolean	This property will be True if the value has "good" quality. If False, detailed information about the quality of the value can be determined from the other properties.
LimitBits	Integer	The limit portion of the code sent by the server.* 0 = The value is free to move up or down. 1 = The value has "pegged" at some lower limit. 2 = The value has "pegged" at some high limit. 3 = The value is constant and cannot move.
Name	String	String representation of the code.*
Quality	Integer	Code that indicates the quality of the value sent by the server.*
VendorBits	Integer	Vendor-specific data within the code.*

*For more information on OPC Quality based on the OPC specifications, refer to **Deconstructing the OPC Quality Field**.

String Definitions

QualityID.Name	QualityID.Code	Description
OPC_QUALITY_BAD	0 (0x00)	Bad quality. Reason unknown.
OPC_QUALITY_COMM_FAILURE	24 (0x18)	Bad quality. Communications have failed and there is no last known value.
OPC_QUALITY_CONFIG_ERROR	4 (0x04)	Bad quality. There is as server configuration problem, such as the item in question has been deleted.
OPC_QUALITY_DEVICE_FAILURE	12 (0x0C)	Bad quality. Device failure detected.
OPC_QUALITY_EGU_EXCEEDED	84 (0x54)	Uncertain quality. The returned value is outside the EGU limits defined for item.
OPC_QUALITY_GOOD	192 (0xC0)	Good quality.
OPC_QUALITY_LAST_KNOWN	20 (0x14)	Bad quality. Communications have failed but there is a last known value available.
OPC_QUALITY_LAST_USABLE	68 (0x44)	Uncertain quality. A data source has not provided the server with a data update within the expected time period. The last known value is returned.*
OPC_QUALITY_LOCAL_OVERRIDE	216 (0xD8)	Good quality. The value has been overridden. This may indicate that an input has been disconnected and that the returned value has been manually "forced".
OPC_QUALITY_NOT_CONNECTED	8 (0x08)	Bad quality. It has been determined that an input is disconnected, or that no value has been provided by data source yet.
OPC_QUALITY_OUT_OF_SERVICE	28 (0x1C)	Bad quality. The item is off scan, locked, or inactive.
OPC_QUALITY_SENSOR_CAL	80 (0x50)	Uncertain quality. The value has either

		exceeded the sensor's limits (limit bits should be set to 1 or 2), or the sensor is out of calibration (limit bits should be 0).
OPC_QUALITY_SENSOR_FAILURE	16 (0x10)	Bad quality. A sensor failure has been detected. Lth limit bits may provide additional information.
OPC_QUALITY_SUB_NORMAL	88 (0x58)	Uncertain quality. The value is derived from multiple sources, and fewer than the required number are good.
OPC_QUALITY_UNCERTAIN	64 (0x40)	Uncertain quality. No specific reason is known.
OPC_QUALITY_WAITING_FOR_INITIAL_DATA	32 (0x20)	Bad quality. No value has been provided to the server yet.

*This is different from the OPC_QUALITY_LAST_KNOWN quality, which is used when the server is unable to read a value from a device. In this case, a data source has failed to write a value to the server in an unsolicited manner.

ResultID Class

ResultID objects are used to describe the result of an operation on an OPC item, such as read, write, and subscribe. ResultID objects will contain the error code provided by the server, its string representation, and a description of the error code. Each item will have its own ResultID, because requests that contain multiple items may succeed for some items and fail for other items.

Property	Data Type	Description
Code	Integer	The code sent by the server for the particular action.
Description	String	The description of the error. The language depends on the locale.
Name	String	The string representation of the code.
Succeeded	Boolean	This property will be True if the operation was a success for the item. If the operation failed, it will be False and the specific reason for that failure can be determined by examining the other properties.

String Definitions

ResultID.Name	ResultID.Code	Description
OPC_E_BADRIGHTS	-1073479674 (0xC0040006)	The item's access rights do not allow the operation.
OPC_E_BADTYPE	-1073479676 (0xC0040004)	The server cannot convert the data between the specified format and/or requested data type and the canonical data type.
OPC_E_DEADBANDNOTSET	-1073478656 (0xC0040400)	The item's deadband has not been set.
OPC_E_DEADBANDNOTSUPPORTED	-1073478655 (0xC0040401)	The item does not support deadband.
OPC_E_DUPLICATENAME	-1073479668 (0xC004000C)	Duplicate name not allowed.
OPC_E_INVALID_PID	-1073479165 (0xC0040203)	The specified Property ID is not valid for the item.
OPC_E_INVALIDCONFIGFILE	-1073479664 (0xC0040010)	The server's configuration file is an invalid format.
OPC_E_INVALIDCONTINUATIONPOINT	-1073478653 (0xC0040403)	The continuation point is not valid.
OPC_E_INVALIDFILTER	-1073479671 (0xC0040009)	The filter string is not valid.
OPC_E_INVALIDHANDLE	-1073479679 (0xC0040001)	The handle value is not valid.
OPC_E_INVALIDITEMID	-1073479672 (0xC0040008)	The Item ID does not conform to the server's syntax.
OPC_E_NOBUFFERING	-1073478654 (0xC0040402)	The server does not support the buffering of data items that are

		collected at a faster rate than the group update rate.
OPC_E_NOTFOUND	-1073479663 (0xC0040011)	The requested object (such as a public group) was not found.
OPC_E_NOTSUPPORTED	-1073478650 (0xC0040406)	The server does not support the writing of quality and/or timestamp.
OPC_E_PUBLIC	-1073479675 (0xC0040005)	The requested operation cannot be done on a public group.
OPC_E_RANGE	-1073479669 (0xC004000B)	The value is out of range.
OPC_E_RATENOTSET	-1073478651 (0xC0040405)	There is no sampling rate set for the specified item.
OPC_E_UNKNOWNITEMID	-1073479673 (0xC0040007)	The Item ID was refused by the server.
OPC_E_UNKNOWNPATH	-1073479670 (0xC004000A)	The item's access path is not known to the server.
OPC_S_CLAMP	-1073479666 (0x0004000E)	A value passed to write was accepted, but the output was clamped.
OPC_S_DATAQUEUEOVERFLOW	-1073478652 (0xC0040404)	Not every detected change has been returned since the server's buffer reached its limit and had to purge the oldest data.
OPC_S_INUSE	-1073479665 (0xC004000F)	The operation cannot be performed because the object is being referenced.
OPC_S_UNSUPPORTEDRATE	-1073479667 (0xC004000D)	The server does not support the requested data rate but will use the closest available rate.
WIN_CONNECT_E_ADVISELIMIT	-2147220991 (0x80040201)	Advise limit exceeded for this object.
WIN_CONNECT_E_NOCONNECTION	-2147220992 (0x80040200)	The client has no callback registered.
WIN_CONNECT_E_CANNOTCONNECT	-2147220990 (0x80040202)	The client cannot connect.
WIN_DISP_E_TYPEMISMATCH	-2147352571 (0x80020005)	Type mismatch.
WIN_E_FAIL	-2147467259 (0x80004005)	Unknown error.
WIN_E_INVALIDARG	-2147024809 (0x80070057)	An invalid parameter was passed to a method call.
WIN_RPC_S_CALL_FAILED	-2147023170 (0x800706BE)	Remote procedure call failed.
WIN_RPC_S_SERVER_UNAVAILABLE	-2147023174 (0x800706BA)	The RPC server is currently not available.
WIN_S_FALSE	1 (0x00000001)	The function was partially successful.
WIN_S_OK	0 (0x00000000)	The operation succeeded.

UserIdentityToken Class

This is the base class for the UserIdentity Tokens.

Property	Data Type	Description
TokenType	UserTokenType	The type of UserIdentity token.

UserIdentityTokenCertificate Class

The UserIdentityTokenCertificate Class defines the UserIdentityToken that will pass an X509v3 certificate for User Authentication.

Property	Data Type	Description
Certificate	Byte	The X509 Certificate in DER format.
PrivateKey	Byte	The PrivateKey for the X509 Certificate in PEM format.
TokenType	UserTokenType	The type of UserIdentity token.

UserIdentityTokenIssuedToken Class

The UserIdentityTokenIssuedToken Class defines the UserIdentityToken for authentication based on a WS-Security compliant token (such as a Kerberos token).

Property	Data Type	Description
EncryptionAlgorithm	String	The encryption algorithm used to encrypt the token data. If the string is empty, the token data is not encrypted.
TokenData	Byte	The XML representation of the token encoded to a Byte string. This token may be encrypted with the server certificate.
TokenType	UserTokenType	The type of UserIdentity token.

UserIdentityTokenUserPassword Class

The UserIdentityTokenUserPassword Class defines the user name and password used with User Authentication.

Property	Data Type	Description
Password	String	This sets the password that will be used for User Authentication.
TokenType	UserTokenType	The type of UserIdentity token.
Username	String	This sets the user name that will be used for User Authentication.

BrowseFilter Enumeration

The BrowseFilter Enumeration is used to specify the type of child elements returned by the Browse method.

Name	Value	Description
ALL	0	All elements will be returned.
BRANCH	1	Only elements of type Branch will be returned.
ITEM	2	Only elements of type Item will be returned.

Property ID Enumeration

The values shown below are the enumeration of all the Item Property ID values.

Name	Value	Description
ACCESSRIGHTS	5	The item's access rights.
ALARM_AREA_LIST	302	The list of alarm areas.
ALARM_QUICK_HELP	301	The alarm's quick help.
CLOSELABEL	106	The item's close label.
CONDITION_LOGIC	304	The item's condition logic.
CONDITION_STATUS	300	The item's condition status.
CONSISTENCY_WINDOW	605	The item's consistency window.
DATA_FILTER_VALUE	609	The value of the item's data filter.
DATATYPE	1	The item's data type.
DEADBAND	306	The item's deadband.
DESCRIPTION	101	The item's description.
DICTIONARY	603	The item's dictionary.
DICTIONARY_ID	601	The dictionary's ID.

ENGINEERINGUNITS	100	Engineering units.
EUINFO	8	The engineering units' information.
EUTYPE	7	The type of engineering units.
HI_LIMIT	308	The item's hi limit.
HIGHEU	102	The high engineering units.
HIGHIR	104	
HIHI_LIMIT	307	
LIMIT_EXCEEDED	305	The item exceeded the limit.
LO_LIMIT	309	The item's lo limit.
LOLO_LIMIT	310	
LOWEU	103	The low engineering units.
LOWIR	105	
OPENLABEL	107	The item's open label.
PRIMARY_ALARM_AREA	303	The area of the primary alarm.
QUALITY	3	The item's quality.
RATE_CHANGE_LIMIT	311	The item's rate change limit.
SCANRATE	6	The item's scan rate.
SOUNDFILE	313	The item's sound file.
TIMESTAMP	4	The item's timestamp.
TIMEZONE	108	The item's time zone.
TYPE_DESCRIPTION	604	The description of the item's type.
TYPE_ID	602	The type of ID.
TYPE_SYSTEM_ID	600	The type of System ID.
UNCONVERTED_ITEM_ID	607	The unconverted item's ID.
UNFILTERED_ITEM_ID	608	The unfiltered item's ID.
VALUE	2	The item's value.
WRITE_BEHAVIOR	606	The item's write behavior.

ServerState Enumeration

Changes in server connection state (as indicated in ServerStateChanged events) may contain one of the enumerated values described in the table below.

Name	Value	Description
CONNECTED	4	The server is connected.
DISCONNECTED	1	The server is disconnected.
ERRORSHUTDOWN	2	The server is shutting down.
ERRORWATCHDOG	3	The ClientAce API watchdog has determined that a server connection has failed. ClientAce may attempt to reconnect to the server depending on the options specified when the Connect method was called.
UNDEFINED	0	The server state is not known.

ReturnCode Enumeration

Most ClientAce API methods will return a code that indicates the operation's level of success. The code may take one of the enumerated values described in the table below. An exception will be thrown in the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors).

Name	Value	Description
ITEMANDQUALITYERROR	2	An error was returned during operation for at least one item. The returned quality for at least one item (either the same or different item) was not good. The items can be determined by checking the ResultID and the quality field of the ItemIdentifier array.
ITEMERROR	1	For at least one item, an error was returned during operation. The item can be determined by checking the ResultID of the ItemIdentifier array.
QUALITYNOTGOOD	2	For at least one item, the returned quality was not good. The item can be determined by checking the quality field of the ItemIdentifier

		array.
SUCCEEDED	0	The function returned successfully.
UNSUPPORTEDUPDATERATE	4	The function returned successfully, but the requested update was not supported by the underlying server. The revised update will be returned to the client. *

*This only applies to the Subscribe and SubscriptionModify methods.

UserTokenType Enumeration

The UserTokenType Enumeration identifies the type of User Authentication that will be used when connecting to the server.

Name	Value	Description
ANONYMOUS	0	No User Authentication is used.
CERTIFICATE	2	Authenticate with a User Certificate.
ISSUEDTOKEN	3	Authenticate with an issued token (such as a Kerberos token).
USERNAME	1	Authenticate with a User name and Password.

DaServerMgt Object

The DaServerMgt Object is a class that is used to connect to an OPC server to collect and manage its data. The object supports OPC DA, OPC XML-DA, and OPC UA server connections. For more information on a specific topic, select a link from the list below.

AccessRights Enumerated Values
DataChanged Event
ReadCompleted Event
ServerStateChanged Event
WriteCompleted Event
Browse Method
Connect Method
Disconnect Method
GetProperties Method
Read Method
ReadAsync Method
Subscribe Method
SubscriptionAddItems Method
SubscriptionCancel Method
SubscriptionModify Method
SubscriptionRemoveItems Method
Write Method
WriteAsync Method
IsConnected Property
ServerState Property

Creating DaServerMgt Object

Users must create an instance of DaServerMgt.

```
[Visual Basic]
Dim WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DAServerMgt
```

```
[C#]
DaServerMgt daServerMgt = new Kepware.ClientAce.OpcDaClient.DaServerMgt ();
```

AccessRights Enumerated Values

The enumeration for the OPC DA item access rights are described in the table below.

Value	Constant Name	Description
0	NOTDEFINED	No rights are defined. This is the default state.

1	READONLY	The item is Read Only.
2	READWRITE	The item can be Read and Written.
3	WRITEONLY	The item is Write Only.

DataChanged Event

Event

```
DataChanged(
        ByVal clientSubscription As Integer,
        ByVal allQualitiesGood As Boolean,
        ByVal noErrors As Boolean,
        ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValueCallback
) Handles daServerMgt.DataChanged
```

Properties

A DataChanged Event will occur when the value or quality of one or more items in a subscription changes. To receive the new item values, implement a DataChanged event handler.

Parameter	Functionality
clientSubscription	This is the handle given to the subscription when created with the Subscribe method.
allQualitiesGood	This flag will be set True if all values included in the data changed notification have good quality.
noErrors	This flag will be set True if there are no item errors (as indicated by the ResultID) in the values included in the data changed notification. If this flag is False, all ItemValue. ResultID objects should be examined to determine which items are in error and why.
itemValues	This array contains the value, quality, and timestamp that have changed. The ItemValue elements also contain ResultID objects that are used to indicate possible item-specific errors.

DataChanged Event Handling Sample Code

The DataChanged Event Handling delegate processes the data change updates received by the ClientAce project from the server. DataChanged Events are sent when a subscription is active and the value or quality of an item in the Subscription changes from one poll cycle to the next.

```
Private Sub daServerMgt_DataChanged(ByVal clientSubscription As Integer, ByVal allQualitiesGood
As Boolean, ByVal noErrors As Boolean, ByVal itemValues() As
Kepware.ClientAce.OpcDaClient.ItemValueCallback) Handles daServerMgt.DataChanged

        BeginInvoke(New
Kepware.ClientAce.OpcDaClient.DaServerMgt.DataChangedEventHandler(AddressOf DataChanged),
New Object() {clientSubscription, allQualitiesGood, noErrors, itemValues})
End Sub

Private Sub DataChanged(ByVal clientSubscription As Integer, ByVal allQualitiesGood As Boolean,
ByVal noErrors As Boolean, ByVal itemValues() As
Kepware.ClientAce.OpcDaClient.ItemValueCallback)
    Try
    ~~ Process the call back information here.
    Catch ex As Exception
    ~~ Handle any exception errors here.
    End Try
End Sub
```

Note: Users must forward the callback to the main thread of the application when accessing the GUI directly from the callback. This is recommended even if the application is running in the background.

Adding a DataChanged Event Handler in Visual Basic

1. To start, declare a DaServerMgt object "WithEvents".
2. Next, dim "WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt".
3. Create a subroutine called "Sub 1" to handles the event. Then, create a subroutine or function called "Sub 2" that will act on the event by updating fields or controls within a form, storing values in a cache, and so forth.
4. Within Sub 1, pass the constructor to a new Kepware.ClientAce.OpcDaClient.DaServerMgt.DataChanged Event Handler to the BeginInvoke Method as a parameter.
5. Within the constructor to the Kepware.ClientAce.OpcDaClient.DaServerMgt.DataChanged Event Handler, pass the address of Sub 2.
6. Finally, within the same BeginInvoke Method, pass an array of objects as the last parameter that contain the relevant arguments from the event (which will become the parameters of Sub 2) to Sub 2.

Adding a DataChanged Event Handler in C#

1. Register the event with "DaServerMgt object. daServerMgt.DataChanged += new DAServerMgt.DataChangedEventHandler(DataChanged)".
2. Then, implement the event handler function as desired.

Example Code

```
[Visual Basic]
Try
 Dim itemValue As Kepware.ClientAce.OpcDaClient.ItemValueCallback
 For Each itemValue In itemValues
 If itemValue.ResultID.Succeeded = True Then
 Console.WriteLine( _
 "Item: " & itemValue.ClientHandle & _
 "Value: " & itemValue.Value & _
 "Quality: " & itemValue.Quality.Name & _
 "Timestamp: " & itemValue.TimeStamp)

 Else
 Console.WriteLine("Item error")
 End If
 Next
Catch ex As Exception
 Console.WriteLine("DataChanged exception. Reason: " & ex.Message)
End Try
```

```
[C#]
private void DataChanged (int clientSubscription, bool allQualitiesGood, bool noErrors,
ItemValueCallback[] itemValues)
{
try
 {
 foreach (ItemValueCallback itemValue in itemValues)
 {
 if (itemValue.ResultID.Succeeded)
 {
 Console.WriteLine(
                                                "Item: {0}
                                                Value: {1},
                                                Quality: {2},
                                                Timestamp: {3}",
                                               itemValue.ClientHandle,
                                               itemValue.Value,
```

```
                                        itemValue.Quality.Name,
                                        itemValue.TimeStamp);
 }
 else
 {
 Console.WriteLine("Item error");
 }
 }
 }
catch (Exception ex)
 {
 Console.WriteLine("DataChanged exception. Reason: {0}", ex);
 }
}
```

ReadCompleted Event

Event

```
[Visual Basic]
ReadCompleted( _
        ByVal transactionHandle As Integer, _
        ByVal allQualitiesGood As Boolean, _
        ByVal noErrors As Boolean, _
        ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValueCallback _
) Handles daServerMgt.ReadCompleted
```

Properties

A ReadCompleted Event will occur when the API has completed an asynchronous read request.

Parameter	Functionality
transactionHandle	This is the handle for the read transaction passed to ReadAsync.
allQualitiesGood	This flag will be set True if all values included in the read completed notification have good quality.
noErrors	This flag will be set True if there are no item errors (as indicated by the ResultID) in the values included in the read completed notification. If this flag is False, users should examine all ItemValue.ResultID objects to determine which items are in error and why.
itemValues	This array contains the value, quality, and timestamp of the items specified in the ReadAsync request. The ItemValue elements also contain ResultID objects that are used to indicate possible item-specific errors.

ReadCompleted Event Handling Sample Code

The ReadCompleted Event Handling delegate processes the completion result of an Asynchronous Read request.

```
Private Sub daServerMgt_ReadCompleted(ByVal transactionHandle As Integer, ByVal allQualitiesGood
As Boolean, ByVal noErrors As Boolean, ByVal itemValues() As
Kepware.ClientAce.OpcDaClient.ItemValueCallback) Handles daServerMgt.ReadCompleted

        BeginInvoke(New
Kepware.ClientAce.OpcDaClient.DaServerMgt.ReadCompletedEventHandler(AddressOf
ReadCompleted), New Object() {transactionHandle, allQualitiesGood, noErrors, itemValues})
End Sub

Private Sub ReadCompleted(ByVal transactionHandle As Integer, ByVal allQualitiesGood As Boolean,
ByVal noErrors As Boolean, ByVal itemValues() As
Kepware.ClientAce.OpcDaClient.ItemValueCallback) Handles daServerMgt.ReadCompleted
    Try
    ~~ Process the call back information here.
    Catch ex As Exception
```

```
    ~~ Handle any exception errors here.
    End Try
End Sub
```

Note: Users must forward the callback to the main thread of the application when accessing the GUI directly from the callback. This is recommended even if the application is running in the background.

Adding a ReadCompleted Event Handler in Visual Basic

1. To start, declare a DaServerMgt object "WithEvents".
2. Dim "WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt".
3. Create a subroutine called "Sub 1" to handles the event. Then, create a subroutine or function called "Sub 2" that will act on the event by updating fields or controls within a form, storing values in a cache, and so forth.
4. Within Sub 1, pass the constructor to a new Kepware.ClientAce.OpcDaClient.DaServerMgt.ReadCompleted Event Handler to the BeginInvoke Method as a parameter.
5. Within the constructor to the Kepware.ClientAce.OpcDaClient.DaServerMgt.ReadCompleted Event Handler, pass the address of Sub 2.
6. Finally, within the same BeginInvoke Method, pass an array of objects as the last parameter that contain the relevant arguments from the event (which will become the parameters of Sub 2) to Sub 2.

Adding a ReadCompleted Event Handler in C#

1. Register the event with "DaServerMgt object. daServerMgt.ReadCompleted += new DAServerMgt.ReadCompletedEventHandler(ReadCompleted)".
2. Then, implement the event handler function as desired.

Example Code

```
[Visual Basic]
Try
 Dim itemValue As Kepware.ClientAce.OpcDaClient.ItemValueCallback
For Each itemValue In itemValues
 If itemValue.ResultID.Succeeded = True Then  Console.WriteLine( _
 "Item: " & itemValue.ClientHandle & _
 "Value: " & itemValue.Value & _
 "Quality: " & itemValue.Quality.Name & _
 "Timestamp: " & itemValue.TimeStamp)
 Else
 Console.WriteLine("Item error")
 End If
 Next
Catch ex As Exception
 Console.WriteLine("ReadCompleted exception. Reason: " & ex.Message)
End Try
```

```
[C#]
private void ReadCompleted (int transactionHandle, bool allQualitiesGood, bool noErrors,
ItemValueCallback[] itemValues)
{
 try
 {
 foreach (ItemValueCallback itemValue in itemValues)
 {
 if (itemValue.ResultID.Succeeded)
 {
 Console.WriteLine(
```

```
                                                    "Item: {0}
                                                    Value: {1},
                                                    Quality: {2},
                                                    Timestamp: {3}",
 itemValue.ClientHandle,
 itemValue.Value,
 itemValue.Quality.Name,
 itemValue.TimeStamp);
 }
 else
 {
 Console.WriteLine("Item error");
 }
 }
 }
 catch (Exception ex)
 {
 Console.WriteLine("ReadCompleted exception. Reason: {0}", ex);
 }
}
```

ServerStateChanged Event

Event

```
ServerStateChanged(
        ByVal clientHandle As Integer,
        ByVal state As Kepware.ClientAce.OpcDaClient.ServerState
) Handles daServerMgt.ServerStateChanged
```

Properties

A ServerStateChanged Event will occur when the API has detected that the connection state with a server has changed. To monitor these changes and take appropriate action in response, implement a ServerStateChanged Event Handler in the client application.

Parameter	Functionality
clientHandle	This is the client handle associated with the particular server connection for a state change notification. This handle is provided by the client though the Connect Method.
state	The current status of the connection.*

*For more information, refer to **ServerState Enumeration**.

ServerStateChanged Event Handling Sample Code

The ServerStateChanged Event Handling delegate processes the change in state messages from the server. These messages indicate if the server is in a run state, is shutting down, and so forth.

```
Private Sub daServerMgt_ServerStateChanged(ByVal clientHandle As Integer, ByVal state As
Kepware.ClientAce.OpcDaClient.ServerState) Handles daServerMgt.ServerStateChanged

        BeginInvoke(New
Kepware.ClientAce.OpcDaClient.DaServerMgt.ServerStateChangedEventHandler(AddressOf
ServerStateChanged), New Object() {clientHandle, state})
End Sub

Private Sub ServerStateChanged(ByVal clientHandle As Integer, ByVal state As
Kepware.ClientAce.OpcDaClient.ServerState)
    Try
    ~~ Process the call back information here.
    Catch ex As Exception
    ~~ Handle any exception errors here.
```

```
    End Try
End Sub
```

Note: Users must forward the callback to the main thread of the application when accessing the GUI directly from the callback. This is recommended even if the application is running in the background.

Adding a ServerStateChanged Event Handler in Visual Basic

1. To start, declare a DaServerMgt object "WithEvents".
2. Next, dim "WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt".
3. Create a subroutine called "Sub 1" to handles the event. Then, create a subroutine or function called "Sub 2" that will act on the event by updating fields or controls within a form, storing values in a cache, and so forth.
4. Within Sub 1, pass the constructor to a new Kepware.ClientAce.OpcDaClient.DaServerMgt.ServerStateChanged Event Handler to the BeginInvoke Method as a parameter.
5. Within the constructor to the Kepware.ClientAce.OpcDaClient.DaServerMgt.ServerStateChanged Event Handler, pass the address of Sub 2.
6. Finally, within the same BeginInvoke Method, pass an array of objects as the last parameter that contain the relevant arguments from the event (which will become the parameters of Sub 2) to Sub 2.

Adding a ServerStateChanged Event Handler in C#

1. Register the event with "DaServerMgtobject. daServerMgt.ServerStateChanged+= newDAServerMgt.ServerStateChangedEventHandler(ServerStateChanged);".
2. Implement the event handler function as desired.

WriteCompleted Event

Event

```
WriteCompleted(
        ByVal transaction As Integer,
        ByVal noErrors As Boolean,
        ByVal itemResults() As Kepware.ClientAce.OpcDaClient.ItemResultCallback
) Handles daServerMgt.WriteCompleted
```

Properties

A WriteCompleted Event will occur when the API has completed an asynchronous write request.

Parameter	Functionality
transaction	The handle for the read transaction passed to WriteAsync.
noErrors	This flag will be set True if there are no item errors (as indicated by the ResultID) in the items included in the write completed notification. If this flag is False, users should examine all ItemResultCallback. ResultID objects to determine which items are in error and why.
itemResults	This array contains the ClientHandle value and ResultID object for every written item.

WriteCompleted Event Handling Sample Code

The WriteCompleted Event Handling delegate processes the completion result of an Asynchronous Write request. The items written in an Asynchronous Write request do not have to be in an Active Subscription.

```
Private Sub daServerMgt_WriteCompleted(ByVal transaction As Integer, ByVal noErrors As Boolean,
ByVal itemResults() As Kepware.ClientAce.OpcDaClient.ItemResultCallback) Handles
daServerMgt.WriteCompleted

        BeginInvoke(New
Kepware.ClientAce.OpcDaClient.DaServerMgt.WriteCompletedEventHandler(AddressOf
```

```
WriteCompleted), New Object() {transaction, noErrors, itemResults})
End Sub

Private Sub WriteCompleted(ByVal transaction As Integer, ByVal noErrors As Boolean, ByVal
itemResults() As Kepware.ClientAce.OpcDaClient.ItemResultCallback)
    Try
    ~~ Process the call back information here.
    Catch ex As Exception
    ~~ Handle any exception errors here.
    End Try
End Sub
```

Note: Users must forward the callback to the main thread of the application when accessing the GUI directly from the callback. This is recommended even if the application is running in the background.

Adding a WriteCompleted Event Handler in Visual Basic

1. To start, declare a DaServerMgt object "WithEvents".
2. Next, dim "WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt."
3. Create a subroutine called "Sub 1" to handles the event. Then, create a subroutine or function called "Sub 2" that will act on the event by updating fields or controls within a form, storing values in a cache, and so forth.
4. Within Sub 1, pass the constructor to a new Kepware.ClientAce.OpcDaClient.DaServerMgt.WriteCompleted Event Handler to the BeginInvoke Method as a parameter.
5. Within the constructor to the Kepware.ClientAce.OpcDaClient.DaServerMgt.WriteCompleted Event Handler, pass the address of Sub 2.
6. Finally, within the same BeginInvoke Method, pass an array of objects as the last parameter that contain the relevant arguments from the event (which will become the parameters of Sub 2) to Sub 2.

Adding a WriteCompleted Event Handler in C#

1. Register the event with "DaServerMgt object. daServerMgt.WriteCompleted += new DAServerMgt.WriteCompletedEventHandler(WriteCompleted)".
2. Then, implement the event handler function as desired.

Example Code

```
[Visual Basic]
Try
 Dim result As Kepware.ClientAce.OpcDaClient.ItemResultCallback
 For Each result In itemResults
 If result.ResultID.Succeeded = False Then
 Console.WriteLine("Write failed for item: " & _
 result.ClientHandle)
 End If
 Next
Catch ex As Exception
 Console.WriteLine("WriteCompleted exception. Reason: " & ex.Message)
End Try
```

```
[C#]
private void WriteCompleted (int transactionHandle, bool noErrors,
ItemResultCallback[] itemResults)
{
 try
 {
 foreach (ItemResultCallback result in itemResults)
```

```
 {
 if (!result.ResultID.Succeeded)
 {
 Console.WriteLine("Write failed for item: {0}",
 result.ClientHandle);
 }
 }
 }
catch (Exception ex)
 {
 Console.WriteLine("WriteCompleted exception. Reason: {0}", ex);
 }
}
```

Browse Method

Method

```
Browse (
        ByVal itemName As String,
        ByVal itemPath As String,
        ByRef continuationPoint  As String,
        ByVal maxElementsReturned As Integer,
        ByVal browseFilter As Kepware.ClientAce.OpcDaClient.BrowseFilter,
        ByVal propertyIDs() As Integer,
        ByVal returnAllProperties As Boolean,
        ByVal returnPropertyValues As Boolean,
        ByRef browseElements() As Kepware.ClientAce.OpcDaClient.BrowseElement,
        ByRef moreElements As Boolean
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The Browse Method is used to search for tags in the address space of an OPC Server. The address space is usually displayed in a tree structure because it is close to the outline of the items and branches of the internal hierarchical structure of the server itself.

Parameter	Use	Functionality
itemName	Input	This parameter specifies the element (branch) for which all child elements will be obtained. If an empty string is passed, the root level of the server will be browsed.
itemPath	Input	Reserved for future use.
continuationPoint	Output	If the number of returned elements is limited by the client (parameter maxElementsReturned). If the server limits the returned elements to a certain number, this parameter will be provided to specify a reference point for follow up Browse calls regarding this element in the server's hierarchy.*
maxElementsReturned	Input	This parameter can be used to define the maximum number of elements the server should return. If this value is set to 0, all elements will be returned.
browseFilter	Input	The BrowseFilter is used to define the type of elements to be returned. Possible values include all, items, branches.
propertyIDs	Input	This specifies the properties that should be obtained when calling the Browse. They will be returned in the associated BrowseElement. This will be ignored if the returnAllProperties parameter is set to True.
returnAllProperties	Input	If the returnAllProperties flag is set to True, all properties of the items will be obtained automatically. The properties will be returned in the associated BrowseElement.
returnPropertyValues	Input	If the returnPropertyValues flag is set to True, the values of the requested properties will be returned.
browseElements	Output	This array contains all child elements of the element specified in

		ItemName.
moreElements	Output	The moreElements parameter indicates when not all child elements are returned.

*If an OPC server returns a continuation point, the Browse must be called again with the same parameters but using the returned Continuation Point to obtain missing child elements of this node.

Notes:

1. For more information on Return Value: ReturnCode, refer to **ReturnCode Enumeration**. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown.
2. Before the Browse method is called, its parent DaServerMgt object must be connected to an OPC server using the Connect method. Otherwise, a null reference exception will be thrown.

Example Code

These examples show how to browse the entire namespace of the connected server using recursive functions calls. The results are placed in a tree view control named "tvItems".

```
[Visual Basic]
'Our main class
Public Class Main
'Create server management object
Dim daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
Dim tvItems As New System.Windows.Forms.TreeView
Sub MainThread()
' Create root node
tvItems.Nodes.Add("KepServerEx")
Dim rootNode As TreeNode = tvItems.Nodes(0)
'We assume a connection has already taken place with our
'Server Management object and our connection is still valid
' Browse from root
Browse("", rootNode)
End Sub
Private Sub Browse(ByVal branchName As String, ByVal node As TreeNode)
Dim itemName As String
Dim itemPath As String
Dim continuationPoint As String = ""
Dim maxElementsReturned As Integer
Dim browseFilter As Kepware.ClientAce.OpcDaClient.BrowseFilter
Dim propertyIDs() As Integer
Dim returnAllProperties As Boolean
Dim returnPropertyValues As Boolean
Dim browseElements() As Kepware.ClientAce.OpcDaClient.BrowseElement
Dim moreElements As Boolean = True
' Set input parameters
itemName = branchName
itemPath = ""
maxElementsReturned = 0
browseFilter = Kepware.ClientAce.OpcDaClient.BrowseFilter.ALL
propertyIDs = Nothing ' prevent Visual Studio warning
returnAllProperties = True
returnPropertyValues = False
browseElements = Nothing ' prevent Visual Studio warning
' Call Browse API method
Try
While moreElements = True
daServerMgt.Browse(itemName, _
```

```
itemPath, _
continuationPoint, _
maxElementsReturned, _
browseFilter, _
propertyIDs, _
returnAllProperties, _
returnPropertyValues, _
browseElements, _
moreElements)
' Handle results
Dim numberOfElementsReturned As Integer = browseElements.GetLength(0)
Dim element As Integer
For element = 0 To numberOfElementsReturned - 1
' Add item to specified tree node
node.Nodes.Add(browseElements(element).Name)
' Browse for item's children (recursive call!!!)
If browseElements(element).HasChildren Then
itemName = browseElements(element).ItemName
Browse(browseElements(element).ItemName, node.Nodes(element))
End If
Next
End While
Catch ex As Exception
MsgBox("Browse exception: " & ex.Message)
End Try
End Sub
End Class
```

```
[C#]
//Our Main Class
public class Main
{
//Create our Server Management Object
DaServerMgt daServerMgt = new Kepware.ClientAce.OpcDaClient.DaServerMgt();

System.Windows.Forms.TreeView tvItems = new System.Windows.Forms.TreeView();
public void BeginBrowse()
{
// Create root node
tvItems.Nodes.Add("KepServerEx");
TreeNode rootNode = tvItems.Nodes[0];
//We assume a connection has already taken place by our
//Server Management object and our connection is still valid
// Browse from root
Browse("", rootNode);
}
private void Browse(string branchName, TreeNode node)
{
// Declare parameters
string itemName;
string itemPath;
string continuationPoint = "";
int maxElementsReturned;
BrowseFilter browseFilter;
int[] propertyIDs = null;
bool returnAllProperties;
bool returnPropertyValues;
```

```
BrowseElement[] browseElements = null;
bool moreElements = true;
// Set input parameters
itemName = branchName;
itemPath = "";
maxElementsReturned = 0;
browseFilter = BrowseFilter.ALL;
returnAllProperties = true;
returnPropertyValues = false;
// Call Browse API method
try
{
while (moreElements == true)
{
daServerMgt.Browse(itemName, itemPath, ref continuationPoint,
maxElementsReturned, browseFilter, propertyIDs,
returnAllProperties, returnPropertyValues, out browseElements, out
moreElements);
// Handle results
int numberOfElementsReturned = browseElements.GetLength(0);
int element;
for (element = 0; element < numberOfElementsReturned; element++)
{
// Add item to specified tree node
node.Nodes.Add(browseElements[element].Name);
// Browse for item's children (recursive call!!!)
if (browseElements[element].HasChildren)
{
itemName = browseElements[element].ItemName;
Browse(browseElements[element].ItemName,node.Nodes[element]);
}
}
}
}
catch (Exception ex)
{
Console.WriteLine("Browse exception. Reason: {0}", ex);
}
}
```

Connect Method

Method

```
Connect(
        ByVal url As String,
        ByVal clientHandle As Integer,
        ByRef connectInfo As Kepware.ClientAce.OpcDaClient.ConnectInfo,
        ByRef connectFailed As Boolean
)
```

Properties

The Connect Method establishes a connection with an OPC server.

Parameter	Use	Functionality
URL	Input	The URL of the OPC servers. **Note:** The syntax of the URL that uniquely identifies a server must follow this format (except for OPC XML-DA):

		[OpcSpecification]://[Hostname]/[ServerIdentifier] OpcSpecification: Selects the OPC Specification to be used. • "opcda" for OPC Data Access 2.05A and later (COM). • **Hostname:** Name or IP Address of the machine that hosts the OPC server. For the local machine, "localhost" must be used (127.0.0.1). • **ServerIdentifier:** Identifies the OPC server on the specified host. • OPC DA (COM) ? [ProgID]/[optional ClassID] **Note:** For OPC DA servers, the API will attempt to connect using the ClassID first. If the ClassID is not given (or is found to be invalid), the API will attempt to connect using the ProgID. **OPC DA Example** opcda://localhost/Kepware.KEPServerEX.V5 opcda://127.0.0.1/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397} **OPC XML-DA Example** http://127.0.0.1/Kepware/xmldaservice.asp **OPC UA Example** opc.tcp://127.0.0.1:49320
ClientHandle	Input	The client application can specify a handle to uniquely identify a server connection. The API will return this handle in ServerStateChanged Events.
ConnectInfo	Input/Output	Additional connection options are specified using the connectInfo parameter. For more information, refer to **ConnectInfo Class**.
ConnectFailed	Output	This indicates whether the initial connection to the underlying server failed. It only applies if the retryConnect flag was set in the connect call.

Example Code

```
[Visual Basic]
Imports Kepware.ClientAce.OpcDaClient

Module Module1

Sub Main()
' Declare variables
'Create server management object
Dim daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
Dim url As String = _
"opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397}"
Dim clientHandle As Integer = 1
Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo
connectInfo.LocalId = "en"
connectInfo.KeepAliveTime = 5000
connectInfo.RetryAfterConnectionError = True
connectInfo.RetryInitialConnection = True
connectInfo.ClientName = "Demo ClientAce VB.Net Console Application"
Dim connectFailed As Boolean

Try
' Call Connect API method
daServerMgt.Connect( _
```

```
url, _
clientHandle, _
connectInfo, _
connectFailed)

' Check result
If connectFailed = True Then
Console.WriteLine("Connect failed.")
Else
Console.WriteLine("Connection to Server Succeeded.")
End If

Catch ex As Exception
Console.WriteLine("Connect exception. Reason: " & ex.Message)
End Try

Console.WriteLine("Hit Any Key to Disconnect and Exit:")
Console.Read()

If daServerMgt.IsConnected Then
daServerMgt.Disconnect()
End If
End Sub

End Module
```

```
[C#]
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Kepware.ClientAce.OpcDaClient;

namespace Simple_CS_Console_Application_VS_2010_13
{

class Program
{

static void Main(string[] args)
{
//Initialize the server object
Kepware.ClientAce.OpcDaClient.DaServerMgt DAserver = new
Kepware.ClientAce.OpcDaClient.DaServerMgt();


//Initialize the connection info class
Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = new
Kepware.ClientAce.OpcDaClient.ConnectInfo();
bool connectFailed;
```

```
string url = "opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-
6F3B160F4397}";

int clientHandle = 1;


// Initialize the connect info object data
connectInfo.LocalId = "en";
connectInfo.KeepAliveTime = 5000;
connectInfo.RetryAfterConnectionError = true;
connectInfo.RetryInitialConnection = true;
connectInfo.ClientName = "Demo ClientAce C-Sharp Console Application";


//Try the server connection
try
{
DAserver.Connect(url, clientHandle, ref connectInfo, out connectFailed);
// Check result

if (connectFailed)
{
Console.WriteLine("Connect failed.");
}
else
{
Console.WriteLine("Connected to Server Succeeded.");
}
}
catch(Exception ex)
{
Console.WriteLine("Connect exception. Reason: {0}", ex);
}


Console.WriteLine("Hit Any Key to Disconnect and Exit:");
Console.Read();

if (DAserver.IsConnected)
{
DAserver.Disconnect();
}
}
}
}
```

Notes:

1. The IsConnected property indicates that a client application has successfully called the Connect method. This does not necessarily indicate whether ClientAce is connected to the server. For example, this property would remain True after a connection has failed and ClientAce is in the process of reconnecting. To test the ClientAce to server connection state, use the **ServerState Property**. The server connection state may also be monitored by implementing the **ServerStateChanged Event** handler.

2. It is highly recommended that client applications wait at least 1 second after disconnecting from a server before attempting to connect to that server again.

Disconnect Method

Method

```
Disconnect()
```

Properties

The Disconnect Method releases the connection to the OPC Server. All subscriptions and resources will be freed.

Example Code

```
[Visual Basic]
If daServerMgt.IsConnected = True Then
 daServerMgt.Disconnect()
End If
```

```
[C#]
if (daServerMgt.IsConnected)
        daServerMgt.Disconnect();
```

Get Properties Method

Method

```
GetProperties(
        ByRef itemIdentifiers As Kepware.ClientAce.OpcDaClient.ItemIdentifier,
        ByVal propertyIDs() As Integer,
        ByVal returnAllProperties As Boolean,
        ByVal returnPropertyValues As Boolean,
        ByRef itemProperties() As Kepware.ClientAce.OpcDaClient.ItemProperties,
) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The GetProperties Method is used to obtain the properties of OPC items.

Parameter	Use	Functionality
itemIdentifiers	Input/Output	The array of itemIdentifiers specifies the OPC items whose properties will be obtained.
propertyIDs	Input	The IDs of the properties to be obtained by the GetProperties call. They will be returned in the associated itemProperties element. This will be ignored if the returnAllProperties parameter is set to True.
returnAllProperties	Input	If this flag is set to True, all properties of the items will be obtained automatically. The properties will be returned in the associated itemProperties element.
returnPropertyValues	Input	The property values will be returned if this flag is set to True.
itemProperties	Output	This array contains ItemProperty objects describing the requested properties of the items.

Note: For more information on Return Value: ReturnCode, refer to **ReturnCode Enumeration**. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown.

Example Code

These examples show how to get the access rights and data type properties of a single item "Channel_1.Device_1.Tag_1".

```
[Visual Basic]
' Declare variables
```

```
Dim itemIdentifiers(0) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"
Dim propertyIDs(1) As Integer
propertyIDs(0) = Kepware.ClientAce.OpcDaClient.PropertyID.ACCESSRIGHTS
propertyIDs(1) = Kepware.ClientAce.OpcDaClient.PropertyID.DATATYPE
Dim returnAllProperties As Boolean = False
Dim returnPropertyValues As Boolean = True
Dim itemProperties() As Kepware.ClientAce.OpcDaClient.ItemProperties
itemProperties = Nothing
Try
' Call GetProperties API method
daServerMgt.GetProperties( _
itemIdentifiers, _
propertyIDs, _
returnAllProperties, _
returnPropertyValues, _
itemProperties)
' Handle results
Dim itemProperty As Kepware.ClientAce.OpcDaClient.ItemProperty
For Each itemProperty In itemProperties(0).RequestedItemProperties
Dim propertyDescription As String = itemProperty.Description()
Dim propertyValue As String = itemProperty.Value.ToString()
Console.WriteLine( _
"Property: " & propertyDescription & _
" Value: " & propertyValue)
Next
Catch ex As Exception
Console.WriteLine("GetProperties exception. Reason: " & ex.Message)
End Try
```

```
[C#]
// Declare variables
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[1];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";
int[] propertyIDs = new int[2];
propertyIDs[0] = (int)PropertyID.ACCESSRIGHTS;
propertyIDs[1] = (int)PropertyID.DATATYPE;
bool returnAllProperties = false;
bool returnPropertyValues = true;
ItemProperties[] itemProperties = null;
try
{
// Call GetProperties API method
daServerMgt.GetProperties(ref itemIdentifiers, ref propertyIDs, returnAllProperties, returnPropertyValues,
out itemProperties);
// Handle results
foreach (ItemProperty itemProperty in itemProperties[0].RequestedItemProperties)
{
string propertyDescription = itemProperty.Description;
string propertyValue = itemProperty.Value.ToString();
Console.WriteLine("Property: {0} Value: {1}",
propertyDescription,
propertyValue);
}
}
```

```
catch (Exception ex)
{
Console.WriteLine("GetProperties exception. Reason: {0}", ex);
}
```

Read Method

Method

```
Read (
        ByVal maxAge As Integer,
        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier,
        ByRef itemValues () As Kepware.ClientAce.OpcDaClient.ItemValue
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The Read Method is used to read one or more values from the OPC server. It is strongly recommended that a Subscription be used if the items are read cyclically (and the changed data is received in the DataChanged Event).

Note: The Read Method allows more than one item to be read at a time. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is recommended whenever it is possible.

Parameter	Use	Functionality
maxAge	Input	This specifies whether the server should return a value from cache or from the device for the specified items. If the freshness of the items cached value is within the maxAge, the cache value will be returned. Otherwise, the server will obtain the data from device. The value of maxAge must be in milliseconds.* **Note:** If maxAge is set to 0, the server will always obtain the data from device.
itemIdentifiers	Input/Output	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same items are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	Output	The array itemValues contains Value, Quality, and Timestamp for each item.

*This is only supported for OPC DA 3.0 servers.

Notes:

1. The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR, QUALITYNOTGOOD, or ITEMANDQUALITYERROR), each of the ReturnID objects should be examined to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value:ReturnCode, refer to **ReturnCode Enumeration**.
2. If a particular data type is desired, specify "ItemIdentifier.DataType". Because it is a requested type, it may not be honored. The item's ResultID will indicate if the server was not able to read the item due to an unsupported data type.

Example Code

This example reads two items: "Channel_1.Device_1.Tag_1" and "Channel_1.Device_1.Tag_2".

```
[Visual BAsic]
Imports Kepware.ClientAce.OpcDaClient

Module Module1

Sub Main()
' Declare variables
'Create server management object
Dim daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
'Create a URL for the server connection this could be an OPC DA or UA connection
Dim url As String = _
"opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397}"
'Create a unique id for the client connection
Dim clientHandle As Integer = 1
'Create and configure the connection infomration object and data
Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo
connectInfo.LocalId = "en"
connectInfo.KeepAliveTime = 5000
connectInfo.RetryAfterConnectionError = True
connectInfo.RetryInitialConnection = True
connectInfo.ClientName = "Demo ClientAce VB.Net Console Application"
Dim connectFailed As Boolean

'Create variable for keyboard input
Dim cki As ConsoleKeyInfo

'Create an OPC Item Identifier Object and Data
Dim OPCItems(0) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
Dim itemValues As Kepware.ClientAce.OpcDaClient.ItemValue()
Dim maxAge As Integer = 0

OPCItems(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
OPCItems(0).ItemName = "Channel1.Device1.Tag1" 'This is a ramping tag from the simdemo project
OPCItems(0).ClientHandle = 1
OPCItems(0).DataType = System.Type.GetType("System.UInt16") 'Set the type to VT_Empty and let
the server return the revised type
Try
' Call Connect API method
daServerMgt.Connect( _
url, _
clientHandle, _
connectInfo, _
connectFailed)

' Check result
If connectFailed = True Then
Console.WriteLine("Connect failed.")
Else
Console.WriteLine("Connection to Server Succeeded.")
End If
```

```
Catch ex As Exception
Console.WriteLine("Connect exception. Reason: " & ex.Message)
End Try

'Display a console prompt
If connectFailed = True Then
Console.WriteLine(vbCrLf & "Hit 'Q' to close console")
Else
Console.WriteLine(vbCrLf & "Hit 'R' to Read a value from the server" & vbCrLf & "Hit 'Q' to
Disconnect and Exit.")
End If

Do
Console.WriteLine(vbCrLf & "Ready:")
cki = Console.ReadKey()

If cki.Key = ConsoleKey.R Then
'Try to read the initialized item
Try
daServerMgt.Read(maxAge, OPCItems, itemValues)

'Add code to handle the read response itemvalues

If (OPCItems(0).ResultID.Succeeded And itemValues(0).Quality.IsGood) Then
Console.WriteLine(vbCrLf & "Read value of: {0}", itemValues(0).Value.ToString)
Else
Console.WriteLine(vbCrLf & "Read Failed with resuilt: {0}", OPCItems(0).ResultID.Description)
End If

Catch ex As Exception
'Handle the read exaception
Console.WriteLine("Sync read failed with exception " & ex.Message)
End Try
End If
Loop While cki.Key < > ConsoleKey.Q

If daServerMgt.IsConnected Then
daServerMgt.Disconnect()
End If
End Sub

End Module
```

[C#]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Kepware.ClientAce.OpcDaClient;

namespace Simple_CS_Console_Application_VS_2010_13
{

class Program
{

static void Main(string[] args)
{
//Initialize the server object
Kepware.ClientAce.OpcDaClient.DaServerMgt DAserver = new
Kepware.ClientAce.OpcDaClient.DaServerMgt();


//Initialize the connection info class
Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = new
Kepware.ClientAce.OpcDaClient.ConnectInfo();
bool connectFailed;
string url = "opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-
6F3B160F4397}";

int clientHandle = 1;


// Initialize the connect info object data
connectInfo.LocalId = "en";
connectInfo.KeepAliveTime = 5000;
connectInfo.RetryAfterConnectionError = true;
connectInfo.RetryInitialConnection = true;
connectInfo.ClientName = "Demo ClientAce C-Sharp Console Application";

//Initialize the key input variable
ConsoleKeyInfo cki;

//Create an OPC Identifier Object and Data
Kepware.ClientAce.OpcDaClient.ItemIdentifier[] OPCItems = new
Kepware.ClientAce.OpcDaClient.ItemIdentifier[1];
Kepware.ClientAce.OpcDaClient.ItemValue[] OPCItemValues = null;
int maxAge = 0;

OPCItems[0] = new Kepware.ClientAce.OpcDaClient.ItemIdentifier();
OPCItems[0].ItemName = "Channel1.Device1.Tag1";
OPCItems[0].ClientHandle = 1;
```

```
//Try the server connection
try
{
DAserver.Connect(url, clientHandle, ref connectInfo, out connectFailed);
// Check result

if (connectFailed)
{
Console.WriteLine("Connect failed.");
}
else
{
Console.WriteLine("Connected to Server Succeeded.");
}
}
catch (Exception ex)
{
Console.WriteLine("Connect exception. Reason: {0}", ex);
}

//Display Console Prompt
if (DAserver.IsConnected == false)
{
Console.WriteLine(" \r\nHit 'Q' to close console");
}
else
{
Console.WriteLine("\r\nHit 'R' to Read a value from the server. \r\nHit 'Q' to Disconnect and Exit.");
}

do
{
cki = Console.ReadKey();
if (cki.Key == ConsoleKey.R)
{
try
{ // Call Read API method

DAserver.Read(maxAge, ref OPCItems, out OPCItemValues);

//add code to handle read response item values
if (OPCItems[0].ResultID.Succeeded & OPCItemValues[0].Quality.IsGood)
{
Console.WriteLine("\r\nRead value of: {0}", OPCItemValues[0].Value.ToString());
}
else
{
Console.WriteLine("\r\nRead Failed with resuilt: {0}", OPCItems[0].ResultID.Description);
}
}
catch (Exception ex)
{
Console.WriteLine("Read exception. Reason: {0}", ex);
```

```
}

}
} while (cki.Key != ConsoleKey.Q);

if (DAserver.IsConnected)
{
DAserver.Disconnect();
}
}
}
}
```

ReadAsync Method

Method

```
ReadAsync (
        ByVal transactionHandle As Integer,
        ByVal maxAge As Integer,
        ByRef itemIdentifiers() as Kepware.ClientAce.OpcDaClient.ItemIdentifier
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The ReadAsync Method is used to asynchronously read items of an OPC Server. The read values will be returned in the ReadCompleted Event. It is strongly recommended that a Subscription be used if the items are read cyclically (and the changed data be received in the DataChanged Event).

Note: The ReadAsynch Method allows more than one item to be read at a time. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is recommended whenever it is possible.

Parameter	Use	Functionality
maxAge	Input	This specifies whether the server should return a value from cache or from the device for the specified items. If the freshness of the items cached value is within the maxAge, the cache value will be returned. Otherwise, the server will obtain the data from device. The value of maxAge must be in milliseconds.* **Note:** If maxAge is set to 0, the server will always obtain the data from device.
itemIdentifiers	Input/Output	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
transactionHandle	Input	The API will return the specified handle along with the requested values in a ReadCompleted event. Thus, a ReadCompleted event may be correlated with a particular call to ReadAsync.

*This is only supported for OPC DA 3.0 servers.

Notes:

1. The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR, QUALITYNOTGOOD, or ITEMANDQUALITYERROR), each of the ReturnID objects

should be examined to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value:ReturnCode, refer to **ReturnCode Enumeration**.

2. If a particular data type is desired, specify "ItemIdentifier.DataType". Because it is a requested type, it may not be honored. The item's ResultID will indicate if the server was not able to read the item due to an unsupported data type.

Example Code

These examples read two items: "Channel_1.Device_1.Tag_1" and "Channel_1.Device_1.Tag_2".

```
[Visual Basic]
' Declare variables
Dim transactionHandle As Integer = 0
Dim maxAge As Integer = 0
Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"
itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"
itemIdentifiers(1).ClientHandle = 2 ' Assign unique handle
Dim returnCode As Kepware.ClientAce.OpcDaClient.ReturnCode
Try
' Call ReadAsync API method
returnCode = daServerMgt.ReadAsync( _
transactionHandle, _
maxAge, _
itemIdentifiers)
' Check result
If returnCode <> _
Kepware.ClientAce.OpcDaClient.ReturnCode.SUCCEEDED Then
Console.WriteLine("ReadAsync failed for one or more items")
' Examine ResultID objects for detailed information.
End If
Catch ex As Exception
Console.WriteLine("ReadAsync exception. Reason: " & ex.Message)
End Try
```

```
[C#]
// Declare variables
int transactionHandle = 0;
int maxAge = 0;
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";
itemIdentifiers[0].ClientHandle = 1; // Assign unique handle
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";
itemIdentifiers[1].ClientHandle = 2; // Assign unique handle
ReturnCode returnCode;
try
{
// Call ReadAsync API method
returnCode = daServerMgt.ReadAsync(transactionHandle, maxAge, ref itemIdentifiers);
// Check result
if (returnCode != ReturnCode.SUCCEEDED)
```

```
{
Console.WriteLine("ReadAsync failed for one or more items");
// Examine ResultID objects for detailed information.
}
}
catch (Exception ex)
{
Console.WriteLine("ReadAsync exception. Reason: {0}", ex);
}
```

Subscribe Method

Method

```
Subscribe(
        ByVal clientSubscription As Integer,
        ByVal active As Boolean,
        ByVal updateRate As Integer,
        ByRef revisedUpdateRate As Integer,
        ByVal deadband As Single,
        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier,
        ByRef serverSubscription As Integer
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The Subscribe Method is used to register items for monitoring. The server will continuously scan the subscribed items at the specified update rate and notify the ClientAce API when any item's values or quality changes. The ClientAce API will relay this information to the client application through a DataChanged Event. This relieves the client of having to make continuous calls to Read or ReadAsync to poll a set of items, and can greatly improve the performance of the client application and server.

Note: The server will send an initial update for all items added to an active subscription.

Parameter	Use	Functionality
clientSubscription*	Input	This allows a meaningful handle to be assigned to each subscription. This value will be returned in each DataChanged event. It provides a way to indicate the subscription for which the data update is intended.
active	Input	This is used to create the subscription as active or inactive. The server will scan the items in a subscription only when the subscription is active. The active state may be changed at any time with the **SubscriptionModify Method**. The subscription active state can be used to optimize the application by signaling the server to stop scanning items that are not currently of interest.
updateRate	Input	This allows the rate at which the server scans the subscribed items to be specified. This is a requested rate: the actual update rate will be decided by the server at the time of this call, but can still vary depending on demands on the server and data source. Update rate values must be in milliseconds.
revisedUpdateRate	Output	This returns the update rate set by the OPC server, which can be different from the requested updateRate. The revised update rate will be in milliseconds.
deadband	Input	This specifies the minimum deviation needed for the server to notify the client of a change of value. The deadband is given a percent (0.0–100.0) of the range of the value. The range is given by the EU Low and EU High properties of the item. A deadband of 0.0 will result in the server notifying the client of all changes in the item's value. The Subscribe method will throw an exception if an invalid deadband value is specified.
itemIdentifiers	Input/Output	The array of itemIdentifiers is used to specify the OPC items that should be added to the subscription.

serverSubscription	Output	The API will assign a unique handle for each subscription. This handle is returned through this parameter and should be stored for later use. The server subscription handle must be specified when modifying or canceling a subscription.

*It is up to the developer to ensure that each clientSubscription number is unique.

Notes:

1. The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be added to the subscription and why. The return code will also indicate if the requested update rate is not supported by the server. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumeration**.
2. For the server to return item values with a particular data type, that particular type must be requested with the ItemIdentifier.DataType property. The ResultID will indicate if the server is able to provide the value as the requested type. If the requested type cannot be provided, the values will be sent in their canonical (default) data type.

Example Code

These examples show how to create a new subscription for the two items "Channel_1.Device_1.Tag_1" and "Channel_1.Device_1.Tag_2".

```
[Visual Basic]
' Declare variables
Dim clientSubscription As Integer = 1
Dim active As Boolean = True
Dim updateRate As Integer = 500
Dim revisedUpdateRate As Integer
Dim deadband As Single = 0
Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"
itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"
itemIdentifiers(1).ClientHandle = 2 ' Assign unique handle
Dim serverSubscription As Integer
Try
' Call Subscribe API method
daServerMgt.Subscribe( _
clientSubscription, active, updateRate, _
revisedUpdateRate, deadband, itemIdentifiers, serverSubscription)
' Check results
Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier
For Each item In itemIdentifiers
If item.ResultID.Succeeded = False Then
Console.WriteLine("Subscribe failed for item: " & item.ItemName)
End If
Next
Catch ex As Exception
Console.WriteLine("Subscribe exception. Reason: " & ex.Message)
End Try
```

```
[C#]
// Declare variables
int clientSubscription = 1;
bool active = true;
```

```
int updateRate = 500;
int revisedUpdateRate;
float deadband = 0;
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";
itemIdentifiers[0].ClientHandle = 1; // Assign unique handle
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";
itemIdentifiers[1].ClientHandle = 2; // Assign unique handle
int serverSubscription;
ReturnCode returnCode;
try
{
// Call Subscribe API method
returnCode = daServerMgt.Subscribe(clientSubscription,
active,
updateRate,
out revisedUpdateRate,
deadband,
ref itemIdentifiers,
out serverSubscription);
// Check results
foreach (ItemIdentifier item in itemIdentifiers)
{
if (!item.ResultID.Succeeded)
{
Console.WriteLine("Subscribe failed for item {0}",
item.ItemName);
}
}
}
catch (Exception ex)
{
Console.WriteLine("Subscribe exception. Reason: {0}", ex);
}
```

SubscriptionModify Method

Method

```
SubscriptionModify(
        ByVal serverSubscription As Integer,
        ByVal active As Boolean,
        ByVal updateRate As Integer,
        ByRef revisedUpdateRate As Integer,
        ByVal deadband As Single
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
SubscriptionModify (
        ByVal serverSubscription As Integer,
        ByVal active As Boolean
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
SubscriptionModify (
        ByVal serverSubscription As Integer,
        ByVal updateRate As Integer,
        ByRef revisedUpdateRate As Integer
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
SubscriptionModify (
        ByVal serverSubscription As Integer,
```

```
            ByVal deadband As Single
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The SubscriptionModify Method is used to modify the properties of an existing subscription created with the Subscribe Method. There are three overloads available to change the active UpdateRate and Deadband subscription properties separately.

Parameter	Use	Functionality
serverSubscription	Input	This identifies the subscription within the API. This handle was returned by the Subscribe Method when the subscription was created. The API will throw an exception if an invalid handle is specified.
active	Input	This makes the subscription active or inactive. When the subscription is active, the server will scan the items and provide data change notifications.
updateRate	Input	This specifies the rate at which the server scans the subscribed items. This is a requested rate: the actual update rate will be decided by the server at the time of this call, and can vary depending on demands on the server and data source. Update rate values must be in milliseconds.
revisedUpdateRate	Output	This returns the update rate set by the OPC server, which can be different from the requested updateRate. The revised update rate will be in milliseconds.
deadband	Input	The specifies the minimum deviation needed for the server to notify the client of a change of value. The deadband is given a percent (0.0–100.0) of the range of the value. The range is given by the EU Low and EU High properties of the item. A deadband of 0.0 will result in the server notifying the client of all changes in the item's value. The API will throw an exception if an invalid deadband value is specified.

Note: The return code indicates the overall success of the call. If the code indicates an item-specific error (such as ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be added to the subscription and why. The return code will also indicate if the requested update rate is not supported by the server. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumeration**.

Example Code

These examples modify the properties of an existing subscription that was created with the Subscribe Method.

```
[Visual Basic]
' Declare variables
Dim serverSubscription As Integer ' Assign handle return from Subscribe
Dim active As Boolean = True
Dim updateRate As Integer = 1000
Dim revisedUpdateRate As Integer
Dim deadband As Single = 0
Try
' Call SubscriptionModify API method
daServerMgt.SubscriptionModify( _
serverSubscription, _
active, _
updateRate, _
revisedUpdateRate, _
deadband)
Catch ex As Exception
Console.WriteLine("SubscriptionModify exception. Reason: " & _
ex.Message)
End Try
```

```
[C#]
// Declare variables
int serverSubscription = 0; // Assign handle return from Subscribe
bool active = true;
int updateRate = 1000;
int revisedUpdateRate;
float deadband = 0;
try
{
// Call SubscriptionModify API method
daServerMgt.SubscriptionModify(serverSubscription,
active,
updateRate,
out revisedUpdateRate,
deadband);
}
catch (Exception ex)
{
Console.WriteLine("An error occurred when attempting to modify the subscription: " + ex.Message);
}
```

SubscriptionAddItems Method

Method

```
SubscriptionAddItems(
        ByVal serverSubscription As Integer,
        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The SubscriptionAddItems Method is used to add items to an existing subscription created with the Subscribe method.

Note: The server will send an initial update for all items added to an active subscription.

Parameter	Use	Functionality
serverSubscription	Input	This identifies the subscription within the API. This handle was returned by the Subscription method when the subscription was created. The API will throw an exception if an invalid handle is specified.
itemIdentifiers	Input/Output	The array itemIdentifiers specifies the OPC items that should be added to the subscription.

Notes:

1. The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be added to the subscription and why. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumeration**.
2. For the server to return item values with a particular data type, that particular type must be requested with the ItemIdentifier.DataType property. The ResultID will indicate if the server is able to provide the value as the requested type. If the requested type cannot be provided, the values will be sent in their canonical (default) data type.

Example Code

These examples add the items "Channel_1.Device_1.Tag_3" and "Channel_1.Device_1.Tag_4" to an existing subscription, created with the Subscribe method.

```
[Visual Basic]
' Declare variables
' Assign handle return from Subscribe
Dim serverSubscription As Integer
Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_3"
itemIdentifiers(0).ClientHandle = 3 ' Assign unique handle
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_4"
itemIdentifiers(1).ClientHandle = 4 ' Assign unique handle
Try
' Call SubscriptionAddItems API method
daServerMgt.SubscriptionAddItems( _
serverSubscription, _
itemIdentifiers)
' Check item results
Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier
For Each item In itemIdentifiers
If item.ResultID.Succeeded = False Then
Console.WriteLine("SubscriptionAddItems failed for item: " & _
item.ItemName)
End If
Next
Catch ex As Exception
Console.WriteLine("SubscriptionAddItems exception. Reason: " & _
ex.Message)
End Try
```

```
[C#]
// Declare variables
// Assign handle return from Subscribe
int serverSubscription = 0;
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_3";
// Assign unique handle
itemIdentifiers[0].ClientHandle = 3;
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_4";
// Assign unique handle
itemIdentifiers[1].ClientHandle = 4;
ReturnCode returnCode;
try
{ // Call SubscriptionAddItems API method
returnCode = daServerMgt.SubscriptionAddItems(serverSubscription,
ref itemIdentifiers);
// Check item results
if (returnCode != ReturnCode.SUCCEEDED)
{
foreach (ItemIdentifier item in itemIdentifiers)
{
if (!item.ResultID.Succeeded)
{
Console.WriteLine("SubscriptionAddItems failed for item: {0}", item.ItemName);
}
}
```

```
}
}
catch (Exception ex)
{
Console.WriteLine("SubscriptionAddItems exception. Reason: {0}", ex);
}
```

SubscriptionCancel Method

Method

```
SubscriptionCancel (
        ByVal serverSubscription As Integer
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The SubscriptionCancel Method is used to cancel an existing subscription created with the Subscribe Method.

Parameter	Use	Functionality
serverSubscription	Input	This identifies the subscription within the API. This handle was returned by the Subscribe Method when the subscription was created. The API will throw an exception if an invalid handle is specified.

Note: In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value: Return Code, refer to **ReturnCode Enumeration**.

Example Code

```
[Visual Basic]
' Declare variables
Dim serverSubscription As Integer ' Assign handle return from Subscribe
Try
 daServerMgt.SubscriptionCancel(serverSubscription)
Catch ex As Exception
 Console.WriteLine("SubscriptionCancel exception. Reason: " & _
 ex.Message)
End Try
```

```
[C#]
// Declare variables
int serverSubscription = 0; // Assign handle return from Subscribe
try
{
 // Call SubscriptionCancel API method
 daServerMgt.SubscriptionCancel(serverSubscription);
}
catch (Exception ex)
{
 Console.WriteLine("SubscriptionCancel exception. Reason: {0}", ex);
}
```

SubscriptionRemoveItems Method

Method

```
SubscriptionRemoveItems (
        ByVal serverSubscription As Integer,
        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier
) As/returns Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The SubscriptionRemoveItems Method removes items from an existing subscription that was created with the Subscribe Method.

Parameter	Use	Functionality
serverSubscription	Input	This identifies the subscription within the API. This handle was returned by the Subscribe Method when the subscription was created. The API will throw an exception if an invalid handle is specified.
itemIdentifiers	Input/Output	The array itemIdentifiers specifies the OPC items that should be removed from the Subscription.

Note: The return code indicates the overall success of the call. If the code indicates an item-specific error (such as ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be removed from the subscription and why. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumeration**.

Example Code

These examples remove the items "Channel_1.Device_1.Tag_1" and "Channel_1.Device_1.Tag_2" from an existing subscription that was created with the Subscribe method.

```
[Visual Basic]
' Declare variables
Dim serverSubscription As Integer ' Assign handle return from Subscribe
Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_3"
itemIdentifiers(0).ClientHandle = 3 ' Assign unique handle
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_4"
itemIdentifiers(1).ClientHandle = 4 ' Assign unique handle
Try
 ' Call SubscriptionRemoveItems API method
 daServerMgt.SubscriptionRemoveItems( _
 serverSubscription, _
 itemIdentifiers)

 ' Check item results
 Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier
 For Each item In itemIdentifiers
 If item.ResultID.Succeeded = False Then
 Console.WriteLine( _
 "SubscriptionRemoveItems failed for item: " & _
 item.ItemName)
 End If
 Next
Catch ex As Exception
 Console.WriteLine("SubscriptionRemoveItems exception. Reason: " & _
 ex.Message)
End Try
```

```
[C#]
// Declare variables
int serverSubscription = 0; // Assign handle return from Subscribe
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_3";
itemIdentifiers[0].ClientHandle = 3; // Assign unique handle
```

```
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_4";
itemIdentifiers[1].ClientHandle = 4; // Assign unique handle
ReturnCode returnCode;
try
{ // Call SubscriptionRemoveItems API method
 returnCode = daServerMgt.SubscriptionRemoveItems(serverSubscription,
                  ref itemIdentifiers);
 // Check item results
 if (returnCode != ReturnCode.SUCCEEDED)
 {
 foreach (ItemIdentifier item in itemIdentifiers)
 {
 if (!item.ResultID.Succeeded)
 {
                                    Console.WriteLine("SubscriptionRemoveItems failed for item: {0}
                                    ", item.ItemName);
 }
 }
 }
}
catch (Exception ex)
{ Console.WriteLine("SubscriptionRemoveItems exception. Reason: {0}", ex); }
```

Write Method

Method

```
Write(
        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier,
        ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValue
) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The Write Method is used to write one or more values to the OPC server.

Note: Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is recommended whenever it is possible.

Parameter	Use	Functionality
itemIdentifiers	Input/Output	The array of itemIdentifiers is used to specify the OPC items that should be written. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	Input	The array itemValues contains the values to be written to the OPC server.

Note: The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors) an exception will be thrown. For more information on Return Value: Return Code, refer to **ReturnCode Enumeration**.

Example Code

These examples write the value "111" to tag "Channel_1.Device_1.Tag_1", and "222" to tag "Channel_1.Device_1.Tag_2".

[Visual Basic]

```
Imports Kepware.ClientAce.OpcDaClient

Module Module1

Sub Main()
' Declare variables
'Create server management object
Dim daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
'Create a URL for the server connection this could be an OPC DA or UA connection
Dim url As String = _
"opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397}"
'Create a unique id for the client connection
Dim clientHandle As Integer = 1
'Create and configure the connection infomration object and data
Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo
connectInfo.LocalId = "en"
connectInfo.KeepAliveTime = 5000
connectInfo.RetryAfterConnectionError = True
connectInfo.RetryInitialConnection = True
connectInfo.ClientName = "Demo ClientAce VB.Net Console Application"
Dim connectFailed As Boolean

'Create variable for keyboard input
Dim cki As ConsoleKeyInfo

'Create an OPC Item Identifier Object and Data
Dim OPCItems(0) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
Dim itemValues As Kepware.ClientAce.OpcDaClient.ItemValue()
Dim OPCWriteValue(0) As Kepware.ClientAce.OpcDaClient.ItemValue
Dim maxAge As Integer = 0

OPCItems(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
OPCItems(0).ItemName = "Channel1.Device1.Tag2" 'This is a ramping tag from the simdemo project
OPCItems(0).ClientHandle = 1
OPCItems(0).DataType = System.Type.GetType("System.UInt16") 'Set the type to VT_Empty and let
the server return the revised type

'Define the randoem varaible for the write value
Dim rndValue As New Random

Try
' Call Connect API method
daServerMgt.Connect( _
url, _
clientHandle, _
connectInfo, _
connectFailed)
```

```
' Check result
If connectFailed = True Then
Console.WriteLine("Connect failed.")
Else
Console.WriteLine("Connection to Server Succeeded.")
End If

Catch ex As Exception
Console.WriteLine("Connect exception. Reason: " & ex.Message)
End Try

'Display a console prompt
If connectFailed = True Then
Console.WriteLine(vbCrLf & "Hit 'Q' to close console")
Else
Console.WriteLine(vbCrLf & "Hit 'R' to Read a value from the server")
Console.WriteLine(vbCrLf & "Hit 'W' to Write a Random Value to the Server.")
Console.WriteLine(vbCrLf & "Hit 'Q' to Disconnect and Exit.")
End If

Do
Console.WriteLine(vbCrLf & "Ready:")
cki = Console.ReadKey()

If cki.Key = ConsoleKey.R Then
'Try to read the initialized item
Try
daServerMgt.Read(maxAge, OPCItems, itemValues)

'Add code to handle the read response itemvalues

If (OPCItems(0).ResultID.Succeeded And itemValues(0).Quality.IsGood) Then
Console.WriteLine(vbCrLf & "Read value of: {0}", itemValues(0).Value.ToString)
Else
Console.WriteLine(vbCrLf & "Read Failed with resuilt: {0}", OPCItems(0).ResultID.Description)
End If

Catch ex As Exception
'Handle the read exception
Console.WriteLine("Sync read failed with exception " & ex.Message)
End Try
End If

If cki.Key = ConsoleKey.W Then
'initialize the value to be written
OPCWriteValue(0) = New Kepware.ClientAce.OpcDaClient.ItemValue
OPCWriteValue(0).Value = rndValue.Next(0, 65535)

'Try to write the value
Try
daServerMgt.Write(OPCItems, OPCWriteValue)

If (OPCItems(0).ResultID.Succeeded) Then
```

```
 Console.WriteLine(vbCrLf & "Write succeeded. Value Written is: {0}", OPCWriteValue
(0).Value.ToString)
 Else
 Console.WriteLine(vbCrLf & "Write Failed with result: {0}", OPCItems(0).ResultID.Description)
 End If
 Catch ex As Exception
 'Handle the write exception
 Console.WriteLine("Sync write failed with exception " & ex.Message)
 End Try
 End If
 Loop While cki.Key <> ConsoleKey.Q

 If daServerMgt.IsConnected Then
 daServerMgt.Disconnect()
 End If
 End Sub

 End Module
```

```
[C#]

 using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using Kepware.ClientAce.OpcDaClient;

 namespace Simple_CS_Console_Application_VS_2010_13
 {

 class Program
 {

 static void Main(string[] args)
 {
 //Initialize the server object
 Kepware.ClientAce.OpcDaClient.DaServerMgt DAserver = new
Kepware.ClientAce.OpcDaClient.DaServerMgt();


 //Initialize the connection info class
 Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = new
Kepware.ClientAce.OpcDaClient.ConnectInfo();
 bool connectFailed;
 string url = "opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-
6F3B160F4397}";

 int clientHandle = 1;


 // Initialize the connect info object data
 connectInfo.LocalId = "en";
```

```
connectInfo.KeepAliveTime = 5000;
connectInfo.RetryAfterConnectionError = true;
connectInfo.RetryInitialConnection = true;
connectInfo.ClientName = "Demo ClientAce C-Sharp Console Application";

//Initialize the key input variable
ConsoleKeyInfo cki;

//Create an OPC Identifier Object and Data
Kepware.ClientAce.OpcDaClient.ItemIdentifier[] OPCItems = new
Kepware.ClientAce.OpcDaClient.ItemIdentifier[1];
Kepware.ClientAce.OpcDaClient.ItemValue[] OPCItemValues = null;
int maxAge = 0;

OPCItems[0] = new Kepware.ClientAce.OpcDaClient.ItemIdentifier();
OPCItems[0].ItemName = "Channel1.Device1.Tag2";
OPCItems[0].ClientHandle = 1;
```

WriteAsync Method

Method

```
WriteAsync(
        ByVal transactionHandle As Integer,
        ByRef itemIdentifiers() As kepware.ClientAce.OpcDaClient.ItemIdentifier,
        ByVal itemValues() As kepware.ClientAce.OpcDaClient.ItemValue
) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

Properties

The WriteAsync Method is used to asynchronously write items to an OPC Server. The write values will be returned in the WriteCompleted Event.

Note: More than one item may be written at a time with the WriteAsync method. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is recommended whenever it is possible.

Parameter	Use	Functionality
transactionHandle	Input	The API will return the specified handle along with the requested values in a WriteCompleted Event. Thus, a WriteCompleted Event can be correlated with a particular call to WriteAsync.
itemIdentifiers	Input/Output	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	Input	The array itemValues contains the Values to be written to the OPC server.

Note: The return code indicates the overall success of the call. If this code indicates an item-specific error (such as ITEMERROR or ITEMANDQUALITYERROR), each of the ReturnID objects should be examined to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumeration**.

Example Code

These examples write the value "111" to tag "Channel_1.Device_1.Tag_1", and "222" to tag "Channel_1.Device_1.Tag_2".

```
[Visual Basic]
' Declare variables
Dim transactionHandle As Integer = 0
Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"
itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"
itemIdentifiers(0).ClientHandle = 2 ' Assign unique handle
Dim itemValues(1) As Kepware.ClientAce.OpcDaClient.ItemValue
itemValues(0) = New Kepware.ClientAce.OpcDaClient.ItemValue
itemValues(0).Value = "111"
itemValues(1) = New Kepware.ClientAce.OpcDaClient.ItemValue
itemValues(1).Value = "222"
Dim returnCode As Kepware.ClientAce.OpcDaClient.ReturnCode
Try
 ' Call WriteAsync API method
  returnCode = daServerMgt.WriteAsync( transactionHandle, itemIdentifiers, _
 itemValues)
 ' Check result
 If returnCode <> _
 Kepware.ClientAce.OpcDaClient.ReturnCode.SUCCEEDED Then
 Console.WriteLine("Write request failed for one or more items")
‘ Examine ResultID objects for detailed information.
 End If
Catch ex As Exception
 Console.WriteLine("WriteAsync exception. Reason: " & ex.Message)
End Try
```

```
[C#]
// Declare variables
int transactionHandle = 0;
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";
itemIdentifiers[0].ClientHandle = 1; // Assign unique handle
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";
itemIdentifiers[1].ClientHandle = 2; // Assign unique handle
ItemValue[] itemValues = new ItemValue[2];
itemValues[0] = new ItemValue();
itemValues[0].Value = "111";
itemValues[1] = new ItemValue();
itemValues[1].Value = "222";
ReturnCode returnCode;
try
{ // Call WriteAsync API method
 returnCode = daServerMgt.WriteAsync(transactionHandle, ref itemIdentifiers, itemValues);
 // Check item results
 if (returnCode != ReturnCode.SUCCEEDED)
 { Console.WriteLine("Write request failed for one or more items");
 // Examine ResultID objects for detailed information.
 }
}
```

```
catch (Exception ex)
{ Console.WriteLine("WriteAsync exception. Reason: {0}", ex); }
```

IsConnected Property

Property

IsConnected As Boolean

Properties

The IsConnected Property is used to check if the client application has successfully called the Connect Method. It does not necessarily indicate whether ClientAce is connected to the server. For example, such a property would remain true even after a connection has failed and ClientAce is in the process of reconnecting. To test the ClientAce to server connection state, use the **ServerState Property**. To monitor the server connection state, implement the **ServerStateChanged Event** handler.

Value	Description
True	The client is connected to ClientAce.
False	The client is not connected to ClientAce.

ServerState Property

Property

ServerStateAs Kepware.ClientAce.OpcDaClient.ServerState

Properties

The ServerState Property is used to determine the status of the server connection.

Value	Description
ServerState*	This describes the current connection state between the ClientAce API and the OPC Server.

*For more information, refer to **ServerState Enumeration**.

ClientAceDA_Junction

The ClientAce DA Junction is a customized .NET control that allows VB.NET or C# programmers to easily link data from OPC DA and OPC UA servers to WinForm controls through a simple drag and drop interface. The ClientAce .NET API is recommended when building advanced custom OPC client applications that require more control over OPC functionality. Features of the ClientAce DA Junction include the following:

- No required detailed knowledge about OPC Data Access interfaces.
- Management of the connection handling procedure for one or multiple OPC servers (including connection establishment, connection monitoring, and reconnection in case of errors).
- Conversion of OPC data from different OPC Data Access interfaces into .NET data types.
- Support for .NET WinForm controls available in Visual Studio and from most Third-Party vendors.

ClientAceDA_Junction Properties

Although these properties can only be set at the time of design, they are accessible as Read Only properties at Runtime.

Property	Data Type	Description
DefaultUpdateRate	Integer	The default update rate set in the DA_Junction Object. This is the update rate used on all items unless overridden in the individual item settings.
ShowTimestampInTooltip	Boolean	This shows the timestamp of the OPC value in the tooltip.
BackColorQualityBad	System Color	The back color of the connected control when the quality of the OPC value is bad.
BackColorError	System Color	The back color of the connected control when the ResultID of the OPC item did not succeed.

DisconnectAllServers Method

This method disconnects all servers in the DA Junction Object.

```
[DisconnectAllServers()
```

ReconnectAllServers Method

This method reconnects all servers in the DA Junction Object.

```
ReconnectAllServers()
```

Example Code

```
[Visual Basic]
Private Sub btnDisconnect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs _
) Handles btnDisconnect.Click
Try
'Disconnects all servers that are currently connected in the DA_Junction
ClientAceDA_Junction1.DisconnectAllServers()
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub
Private Sub btnReconnect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs _
) Handles btnReconnect.Click
Try
'Reconnects all servers that are currently connected in the DA_Junction
ClientAceDA_Junction1.ReconnectAllServers()
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub
```

```
[C#]
private void btnDisconnect_Click(object sender, EventArgs e)
{
try
{
//Disconnects all servers that are currently connected in the DA_Junction
clientAceDA_Junction1.DisconnectAllServers();
}
catch (Exception ex)
{
MessageBox.Show ("Received Exception: " + ex.Message);
}
}
private void btnReconnect_Click(object sender, EventArgs e)
{
try
{
//Reconnects all servers that are currently connected in the DA_Junction
clientAceDA_Junction1.ReconnectAllServers();
}
catch (Exception ex)
{
MessageBox.Show("Received Exception: " + ex.Message);
}
}
```

Project Setup

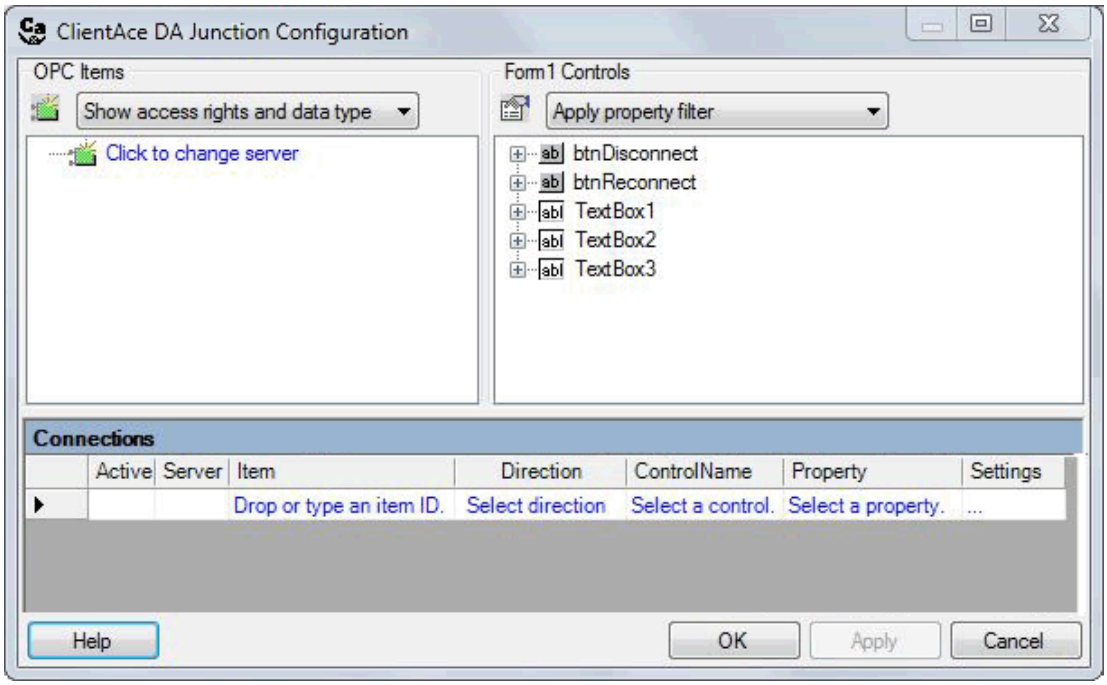
For more information on DA Junction project setup, select a link from the list below.

DA Junction Configuration Window
A Sample Project Using DA Junction with VB.NET or C#
Item Update Rate
Disabling DataChange While the Control Has Focus

DA Junction Configuration Window

The DA Junction Configuration Window is divided into three main panes. To jump to a specific pane, select a link from the list below.

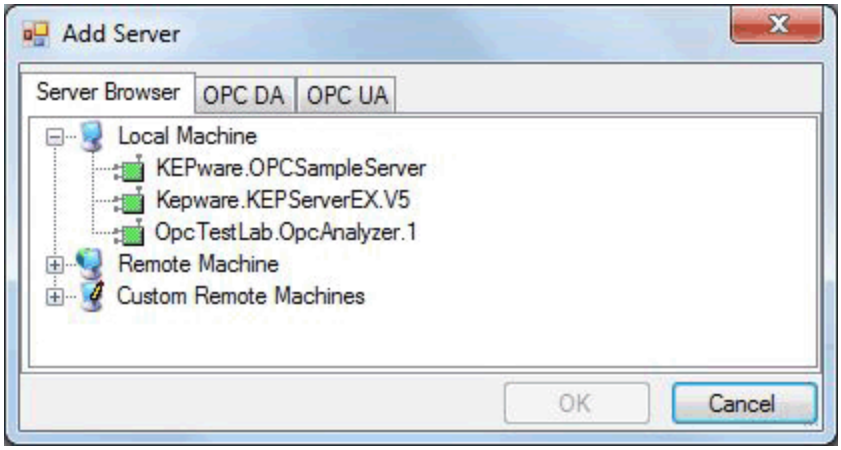
OPC Items
Controls
Connections Pane
Connection Settings



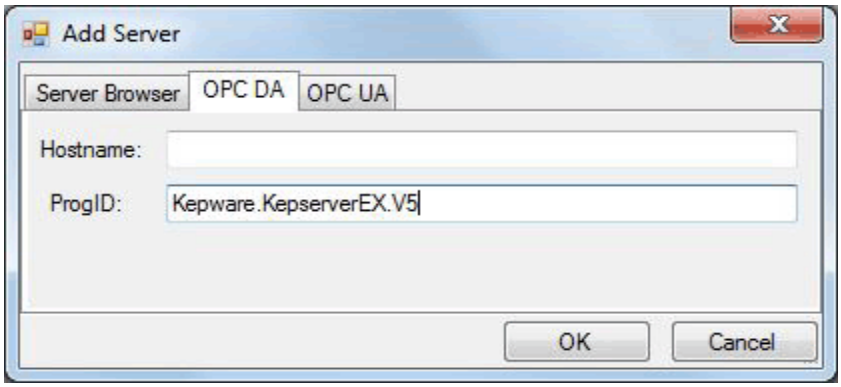
OPC Items

The OPC Items pane displays the items from an OPC server project and shows where the OPC Servers are connected. The DA Junction can connect to OPC DA and OPC UA servers. Users can add servers to this control in three different ways: through the Server Browser, through OPC DA, and through OPC UA.

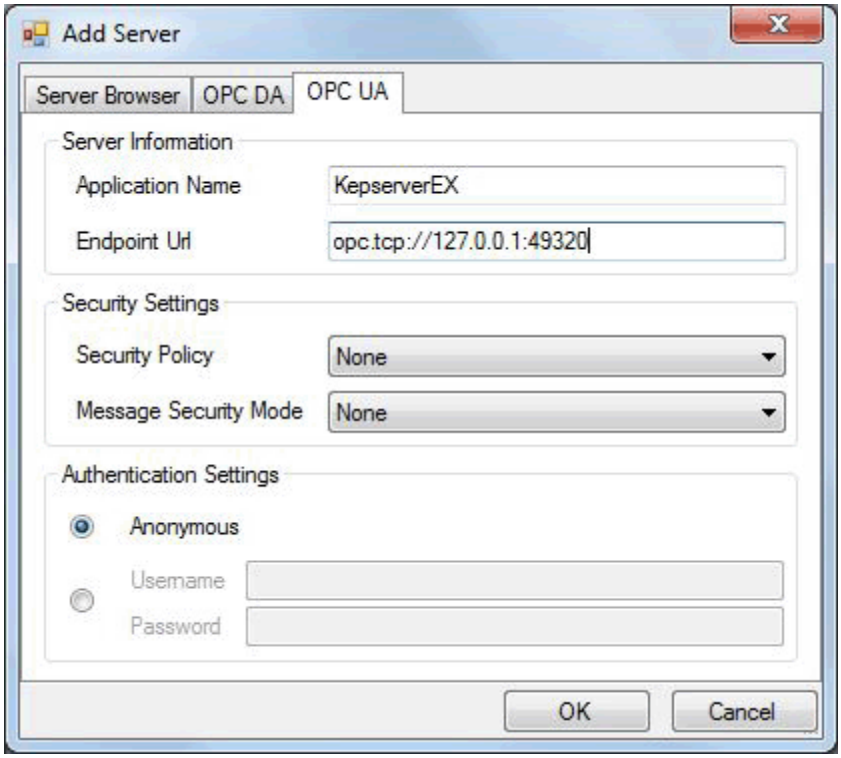
Server Browser
The Server Browser tab allows users to browse for Enumerated OPC DA Servers or OPC UA servers that have been registered to a Local UA Discovery server.



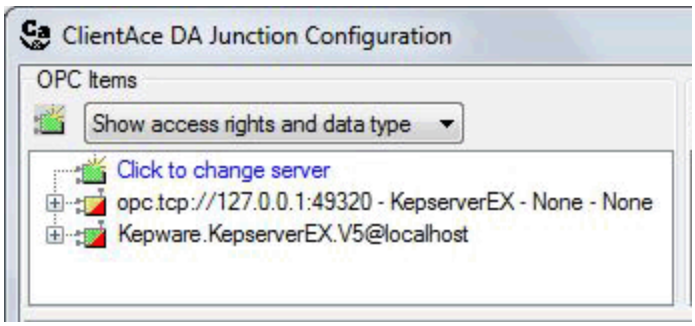
OPC DA
The OPC DA tab allows users to specify a Hostname and ProgramID for servers that may not be enumerated on a PC.



OPC UA
If there are no Discovery Servers available, the Endpoint URL and Application Name can be specified instead. Users can also specify the Security and Authentication settings as configured in the UA Server.

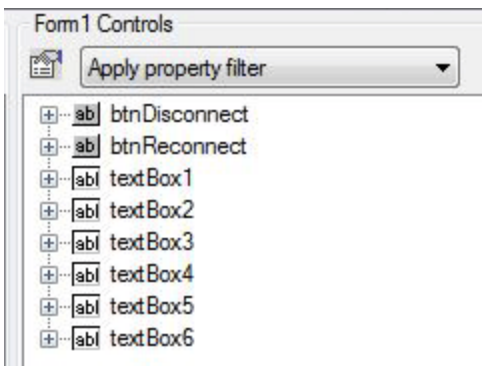


Note: Once connections are made, they will be listed beneath the OPC Items pane. By expanding the server connections, users can view the server folders that contain OPC Items. Selected items will be placed in Connections. For more information, refer to **Connections Pane**.

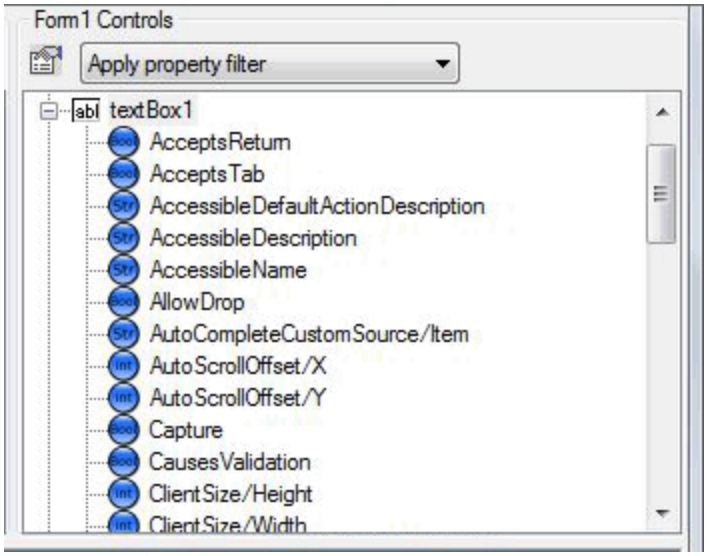


Controls

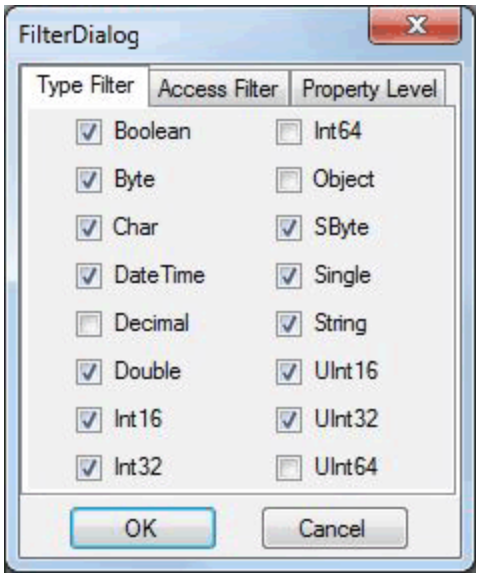
The Controls pane determines which control properties will be displayed. The example below demonstrates the 6 controls on Form1.



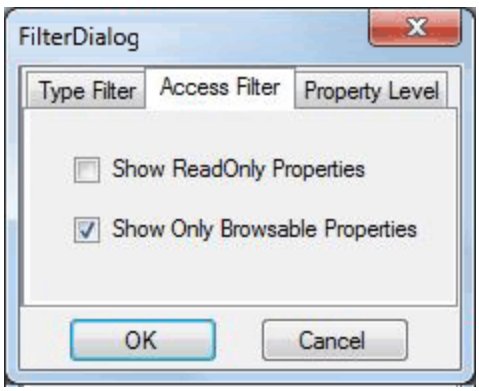
Note: The example below displays the selections for **Show all properties**.



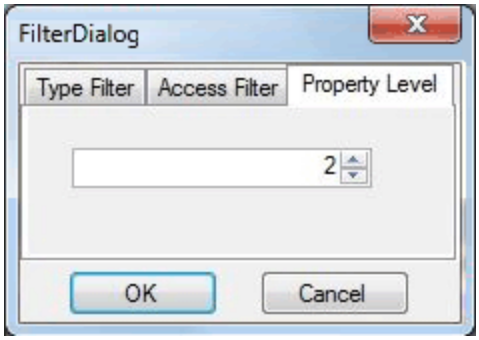
Note: The example below displays the selections for **Apply property filter** located in the **Filter** dialog. The **Type Filter**, which includes a checklist of available data types, is found in the first tab.



Note: The example below displays the selections for **Access Filter tab** located in the **Filter** dialog. The **Show Read Only Properties** field is unchecked by default because data is usually written from the OPC server to the property of the user interface control. To write data from the property, check **Show Read Only Properties** in the OPC server.

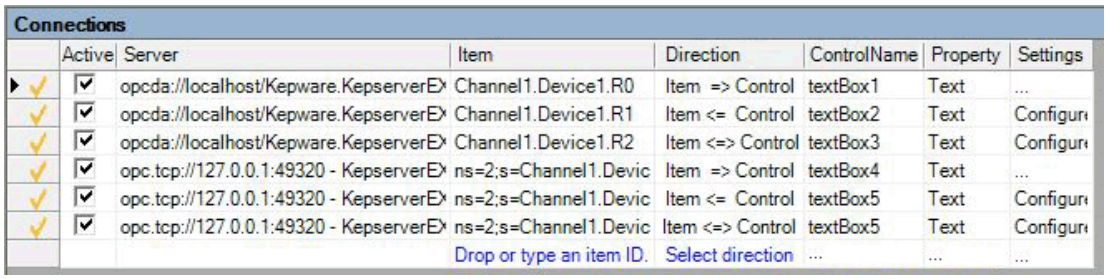


Note: The example below displays the selections for **Property Level** tab in the Filter dialog. The default level is 2. The higher the number is, the greater the level of property detail that will be shown. If the end node of a given item is at level 2, then only 2 levels will be shown for that item if the property level filter is set to 2 or higher. Likewise, if the level filter is set to 3, then only 3 levels of property detail will be shown even if a given item's end node is at level 4 or higher.



Connections Pane

The Connections pane is used to modify the tag state, server name, tag item, and data direction. It can also be used to modify or set Visual Studio controls and properties (and set triggers).



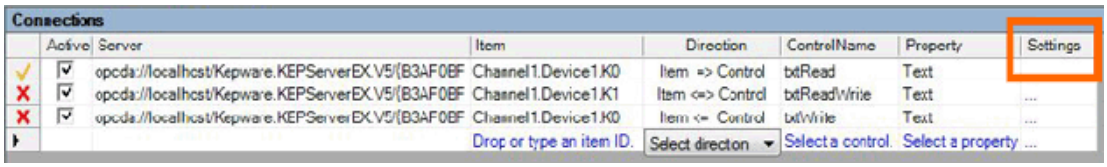
Direction

Direction specifies whether the Visual Studio control is Read Only, Write Only, or Read/Write. The default property is shown in **bold**.

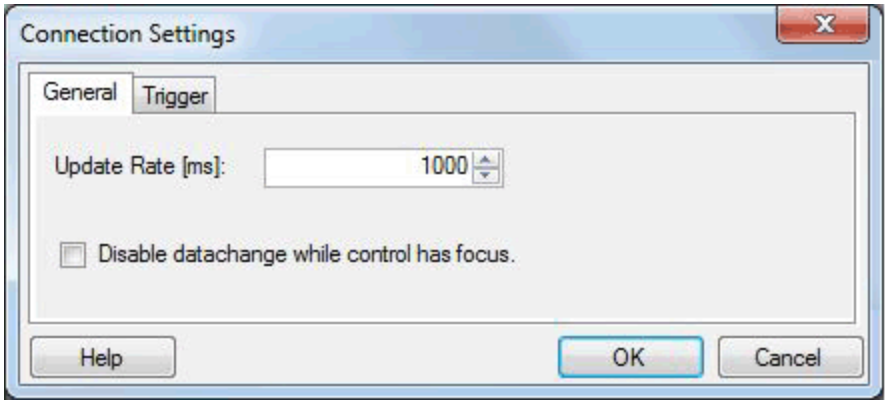
Direction	Property	Description
Item =>Control	**Read Only**	Direction of data is from Item to Control only.
Item <= Control	Write Only	Direction of data is from Control to Item only.
Item <=> Control	Read/Write	Data flows in both directions.

Connection Settings

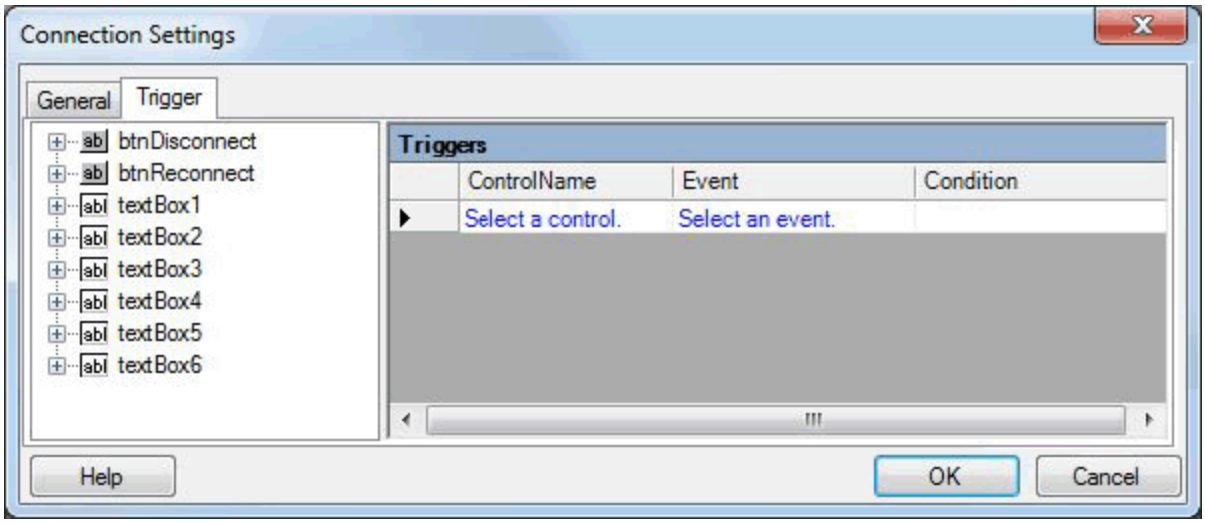
To access an item's Connection Settings, locate the **Settings** column and then click **Browse**.



Note: The Connection Settings window has two tabs: **General** and **Trigger**. The General tab is used to specify the update rate, and determine whether to disable DataChange while the control has focus.



The Trigger tab is used to select the control, browse events, and select an event that will trigger a write to the OPC tag connected to the control. For more information, refer to **Triggers**.



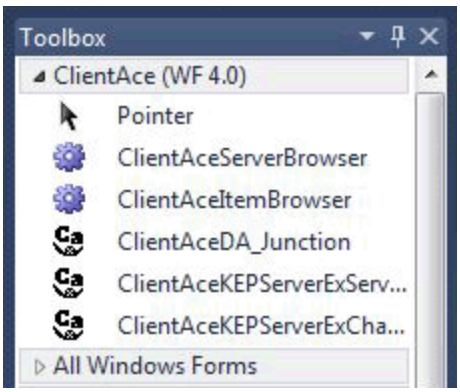
A Sample Project Using DA Junction with VB.NET or C#

Microsoft Visual Studio supports many different Third-Party .NET controls that can be connected to OPC tag items through the Kepware.ClientAce.DA_Junction control library. For more information, refer to the instructions below.

Important: All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

Locating the ClientAceDA Junction Control

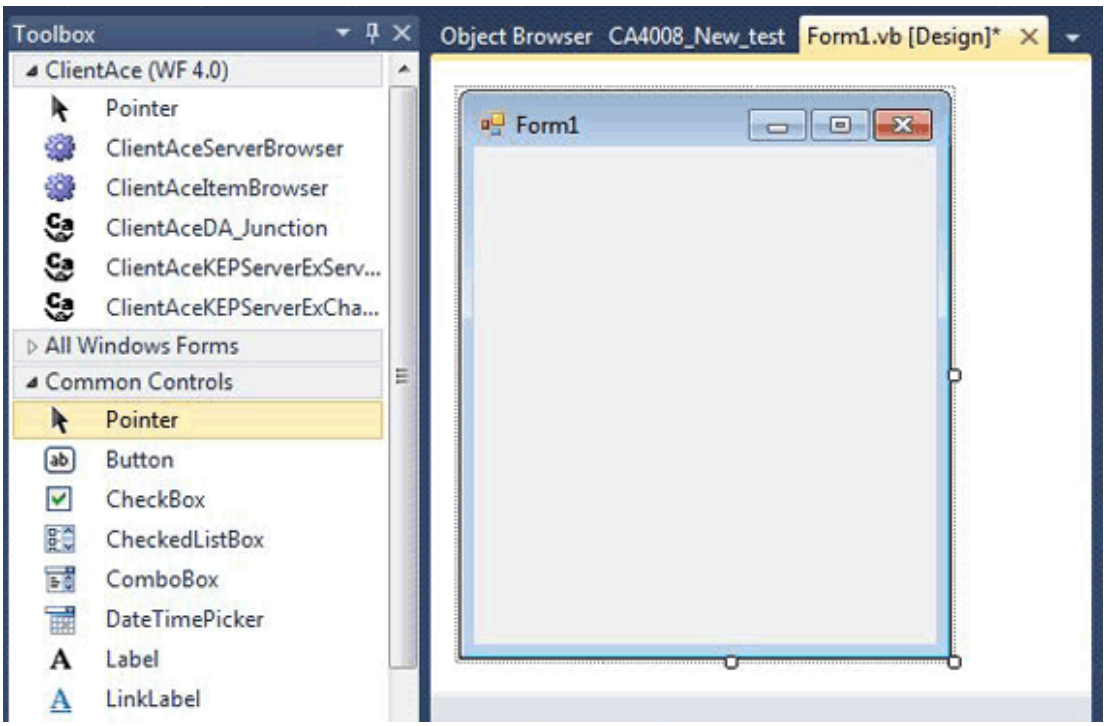
1. To start, open the **Visual Basic Toolbox**, and then locate the **ClientAce** tab. Verify that the **ClientAceDA_Junction** control is listed.



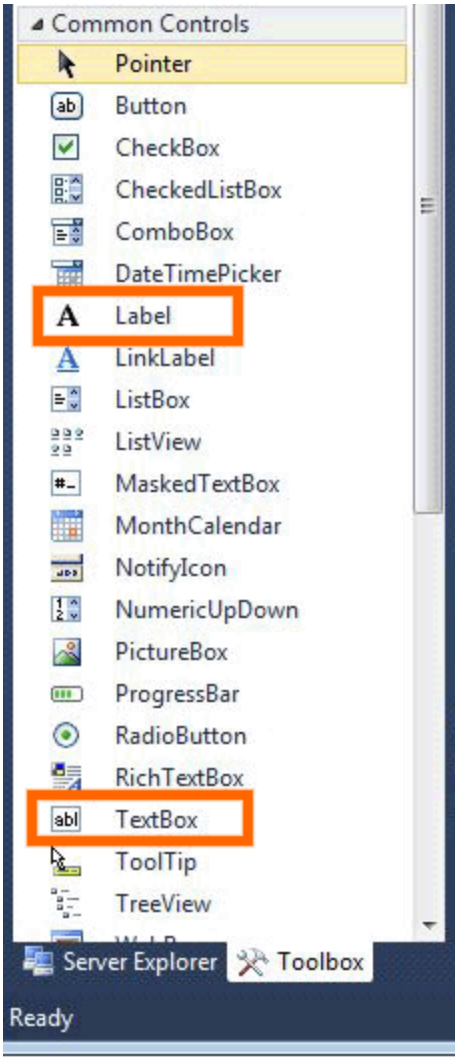
Note: If the ClientAceDA Junction control is missing, refer to **Missing Controls**.

Adding VB/C# Controls

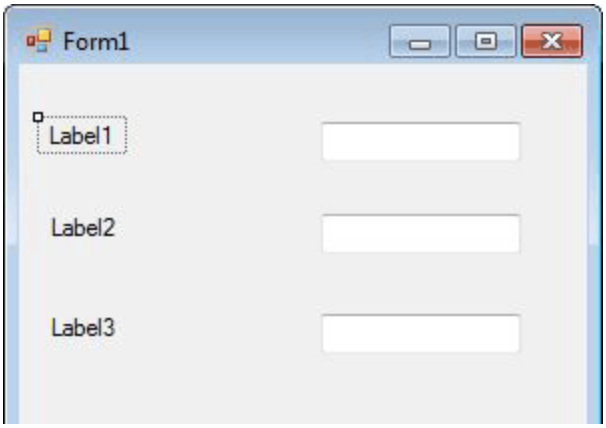
1. Add **VB/C# Controls** to a blank Windows Form.
2. Next, drag and drop the ClientAceDA_Junction control from the Toolbox to the new form. The control label "ClientAceDA_Junction1" will be displayed.



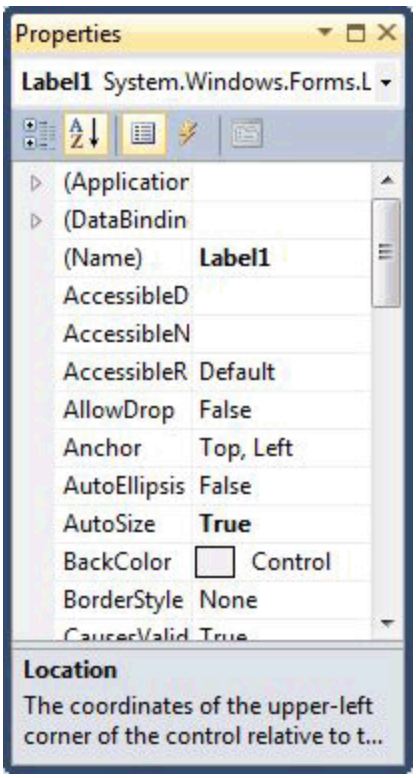
3. Next, drag and drop three **VB/C# Label** controls and three **TextBox** controls onto the form. These controls are located beneath the **Windows Forms** tab in the Toolbox.



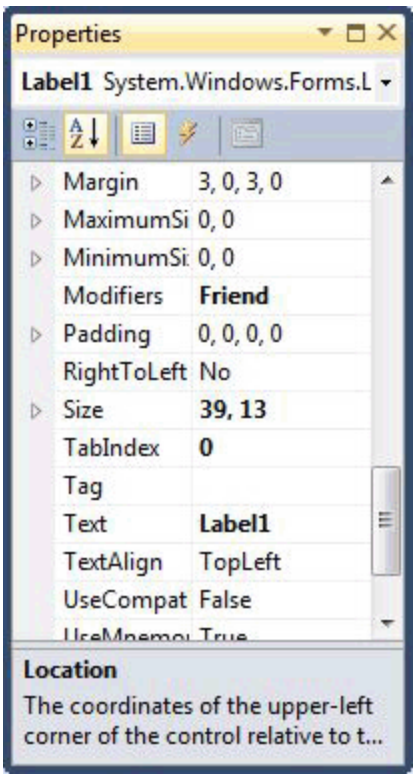
4. To change the controls' names and text properties to be more descriptive, open **Properties** and then click **View**.
5. Next, select **Properties Window** and click once on the **Label1** control to select it.



6. In **Properties**, change the **Name** to "lblRead".



7. Then, change the **Text** property to "ReadVal".



8. Repeat this procedure for all five controls. The changes made to the controls are displayed in the table below.

Default Control Name	New Name Property	New Text Property
Label1	lblRead	ReadVal
Label2	lblWriteValue	WriteVal
Label3	lblReadWriteValue	ReadWriteVal
TextBox1	txtRead	*
TextBox2	txtWrite	*
TextBox3	txtReadWrite	*

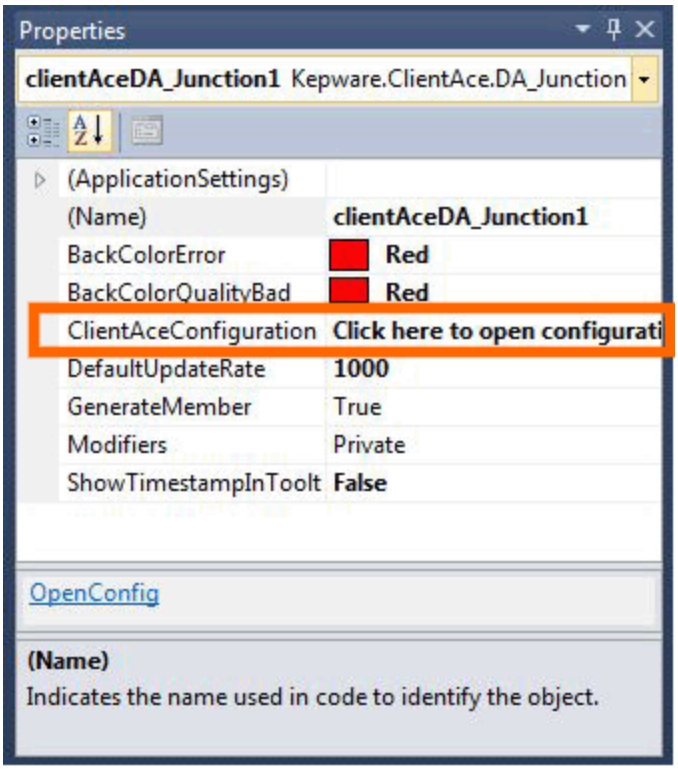
*Leave the Text property for the TextBox controls blank, because they will be updated automatically by the OPC tag items.

Invoking the ClientAce DA Junction Configuration

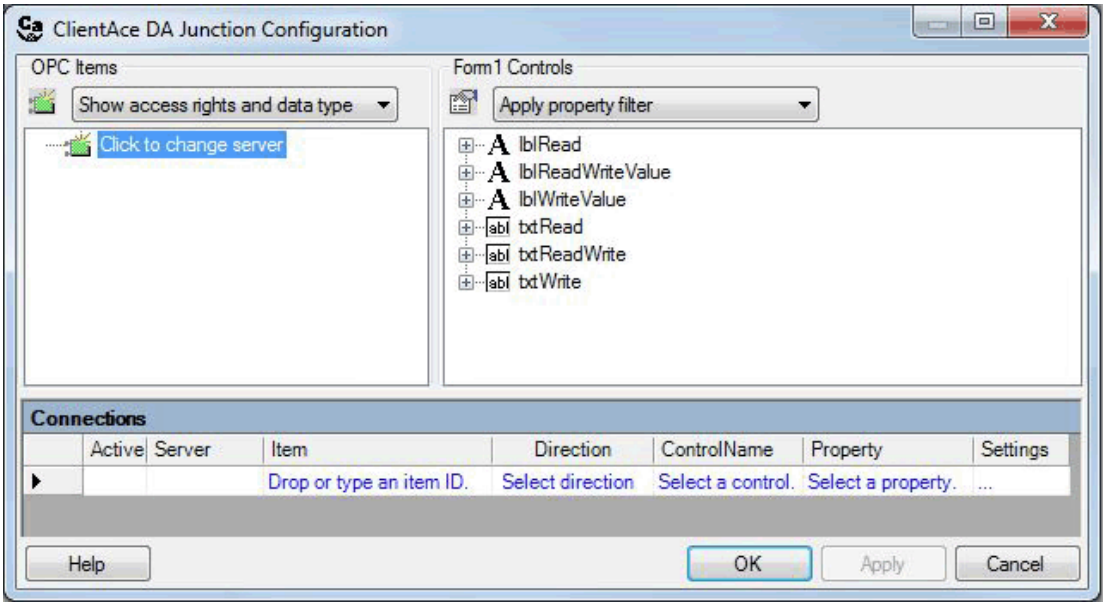
1. Next, invoke the ClientAce DA Junction configuration. Then, click on the ClientAceDA_Junction1 control to select the ClientAceDA_Junction1 property.

   Note: In ClientAce V4.0, users can also open the configuration by double-clicking on the DA Junction Object.

2. In **Properties**, click to select the **ClientAceConfiguration** property. Then, click the **Ellipses** button to launch the **ClientAce DA Junction Configuration** window.

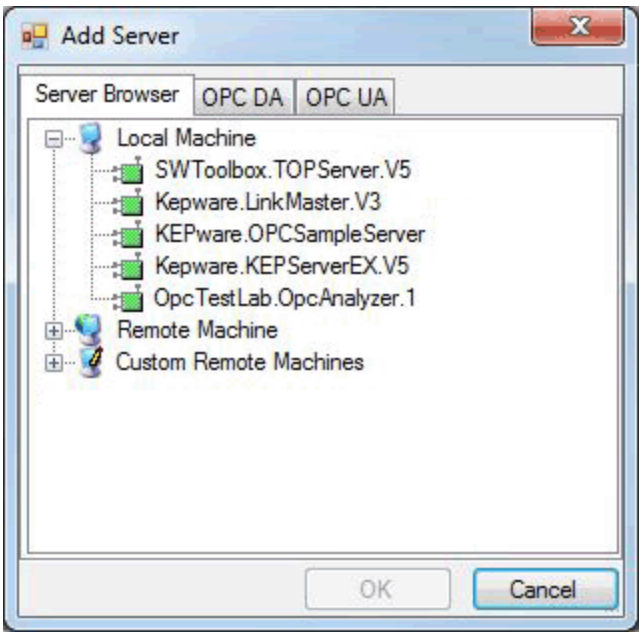


3. Use the **OPC Items** pane to add local and remote servers (and to browse for OPC tag items). To view the VB/C# controls being displayed, open the **Control** pane. For more information, refer to **DA Junction Configuration Window**.

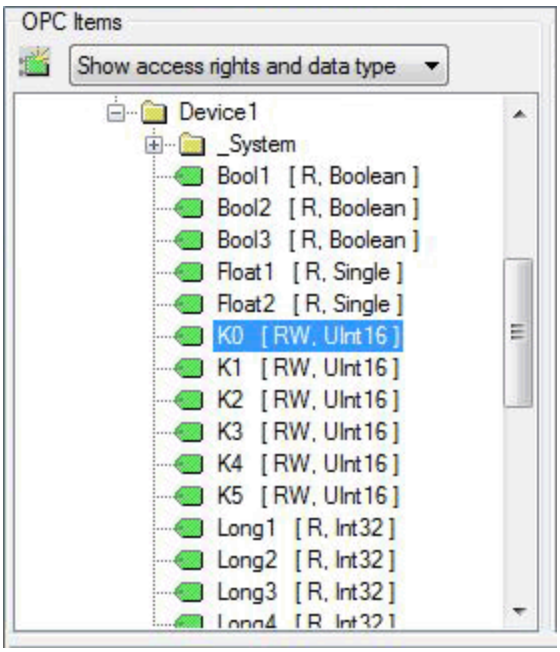


Connecting to OPC Servers and Adding Tags

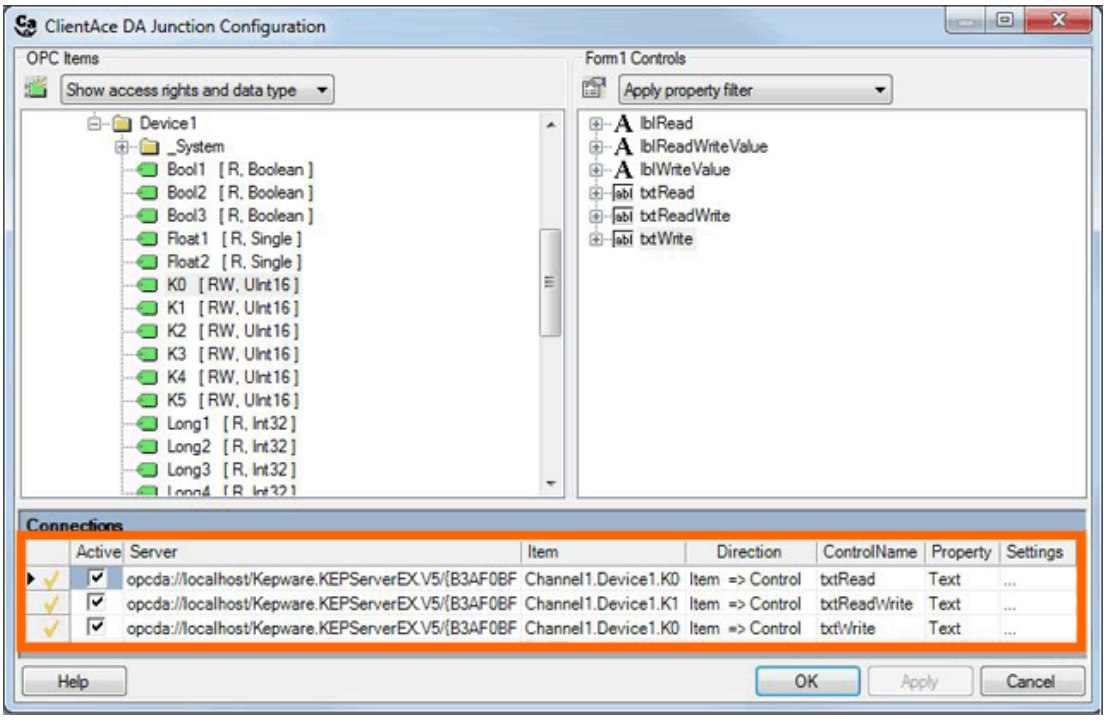
1. Next, double-click on **Click to add a server link** in the left pane of the window.
2. Expand the nodes **Local Machine**, **Remote Machine**, or **Custom Remote Machines** to select the server of interest. In this example, "KEPware.OPCSampleServer" is used.



3. Next, browse the OPC server to locate the tags to which the Visual Studio controls can connect. Then, drag and drop each OPC tag item onto the Visual Studio control.



Note: For example, drag the "K0" Tag to the txtRead and txtWrite controls. Then, drag the K1 Tag to the txtReadWrite textbox control. The tag items should then be listed in the **Connections** grid.



Modifying the Connections Settings

Connections Grid

The Connections grid (located at the bottom of the Configuration Window) is used to modify the tag state, server name, tag item, data direction, Visual Studio controls, properties, and to set triggers. For more information, refer to **DA Junction Configuration Window**.

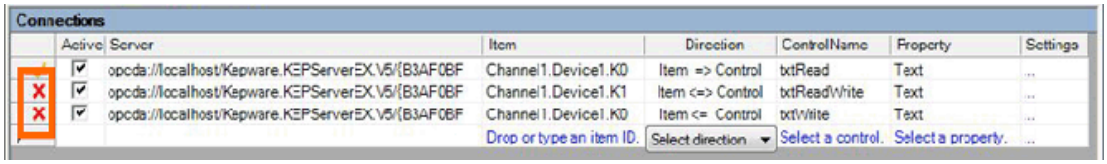
Direction Property

Direction determines whether the Visual Studio control is Read Only, Write Only, or Read/Write. For more information, refer to **Connections Pane**.

For this example, leave the txtRead control at the default Read Only setting. Then, change the txtReadWrite control Read/Write, and the txtWrite control to Write Only. For more information, follow the instructions below.

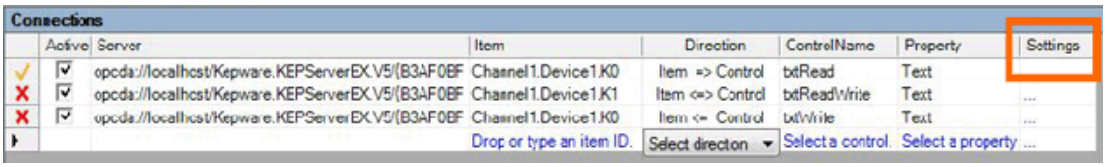
1. For the txtReadWrite control, click the **Direction** column. Then, select **Item <=> Control** from the drop-down menu.
2. For the txtWrite control, click the **Direction** column. Then, select **Item <= Control** from the drop-down menu.

Note: When the direction is changed to Write Only (<=) or Read/Write (<=>), the item will display a red "X" as shown in the image below. The red "X" signifies an error: the control has been set to Write Only or Read/Write but the control does not yet have its write conditions specified. The **Triggers** property specifies the conditions for the write procedures. For more information, refer to "Triggers" below.

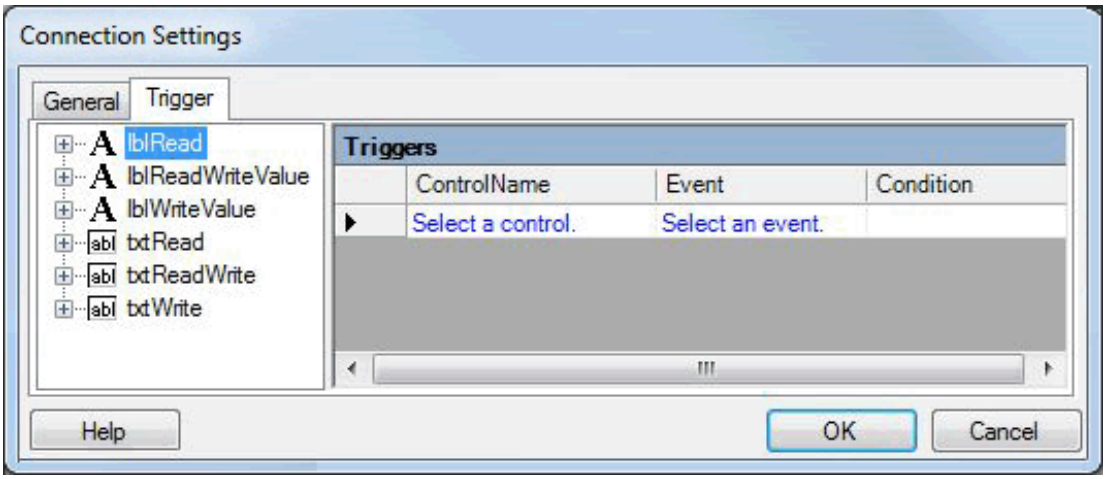


Triggers

1. To access an item's Trigger property, select the **Settings** column. Then, click **Ellipses**.



2. In **Connection Settings**, select the **Trigger** tab.



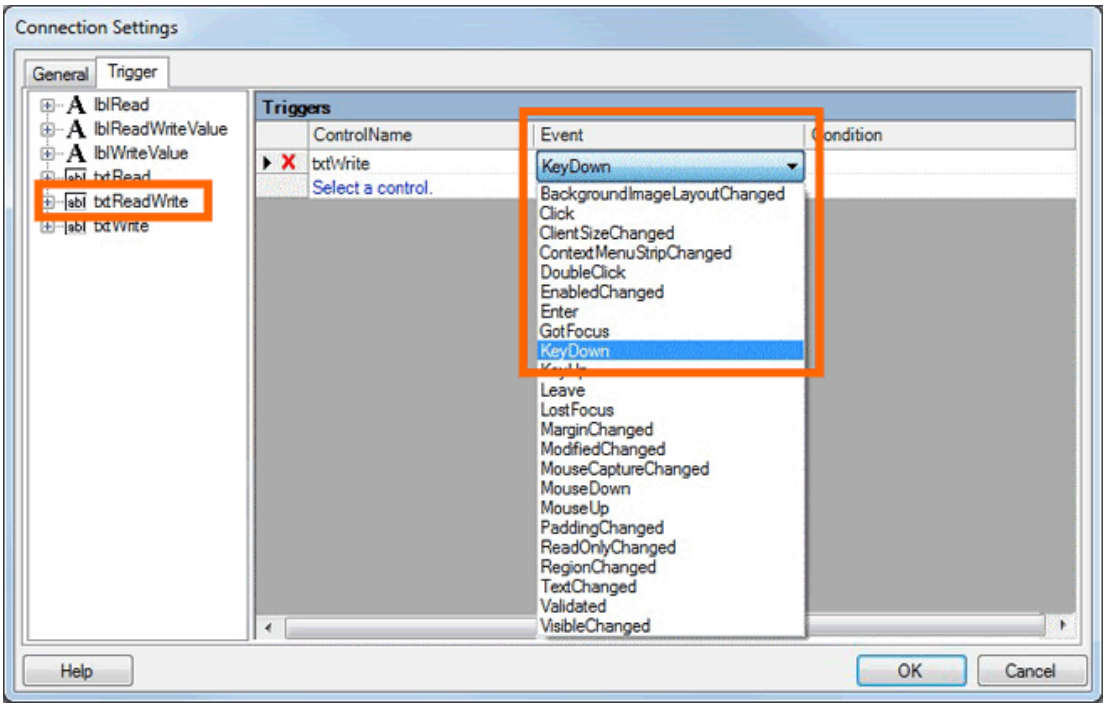
Note: The Trigger tab is used to select the control, to browse events, and to select an event that will trigger a write to the OPC tag connected to the control. For example, the txtReadWrite and txtWrite controls need to have their write conditions specified as follows:

- The txtReadWrite control's KeyDown event triggers writes on the txtReadWrite Visual Studio control.
- The txtWrite control's KeyDown event triggers writes on the txtWrite Visual Studio control.

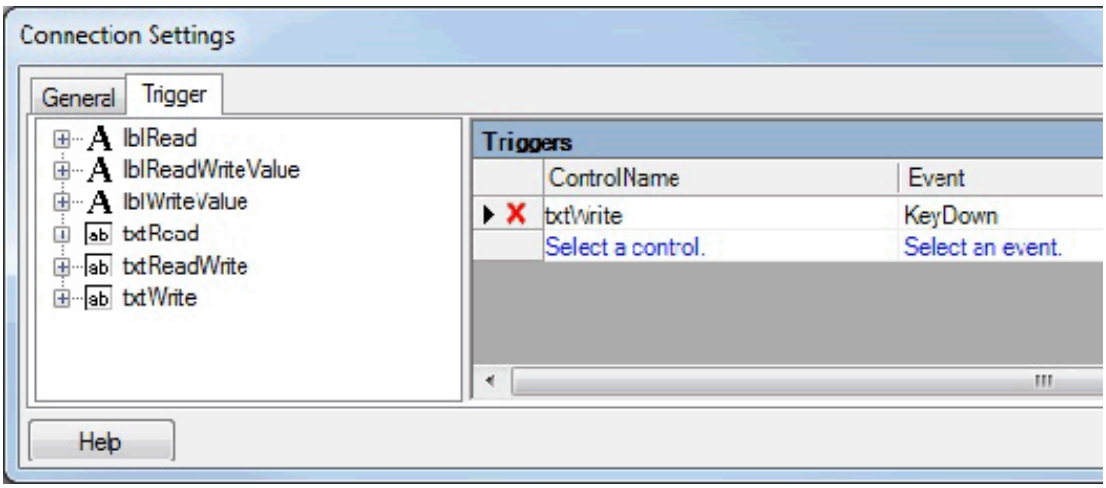
Specifying Write Conditions in the Trigger Tab

The following steps must be performed for both the txtReadWrite and the txtWrite controls.

1. To start, select and expand the txtReadWrite control to display its properties.
2. In the **Trigger** tab, locate the **Event** drop-down list and then select **KeyDown**. Alternatively, drag the KeyDown property and drop it in the Event column.

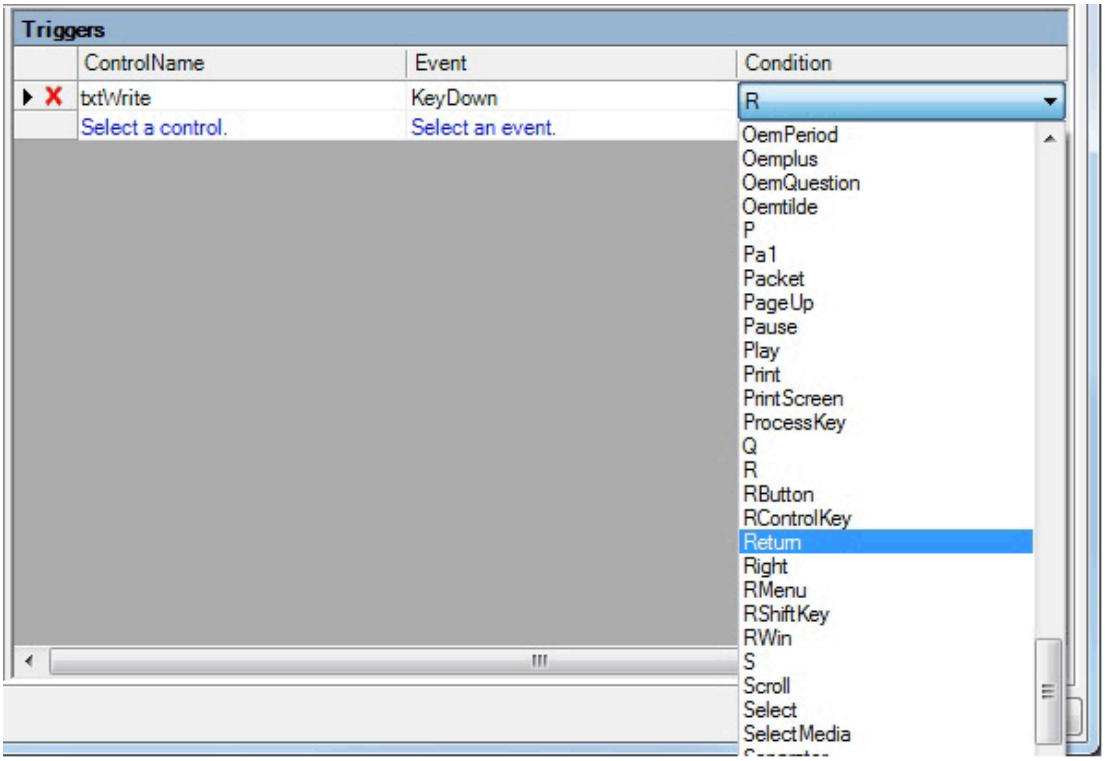


3. Then, click **OK**.
4. Next, return to the Configuration Screen and repeat the process for the txtWrite control.
5. To start, select and expand the txtWrite control to display its properties.
6. In the **Trigger** tab, locate the **Event** drop-down list and then select **KeyDown**. Alternatively, drag the KeyDown property and drop it in the Event column.



7. Then, click **OK**.

Note: When applicable, the **Condition** field will provide a drop-down menu of conditions. For example, if a control is added with KeyDown in the **Event** field, the Condition drop-down menu will display a list of valid keys from which users can choose.



8. To save the changes, click **OK** at the bottom of the Configuration screen. Then, build and run the application.

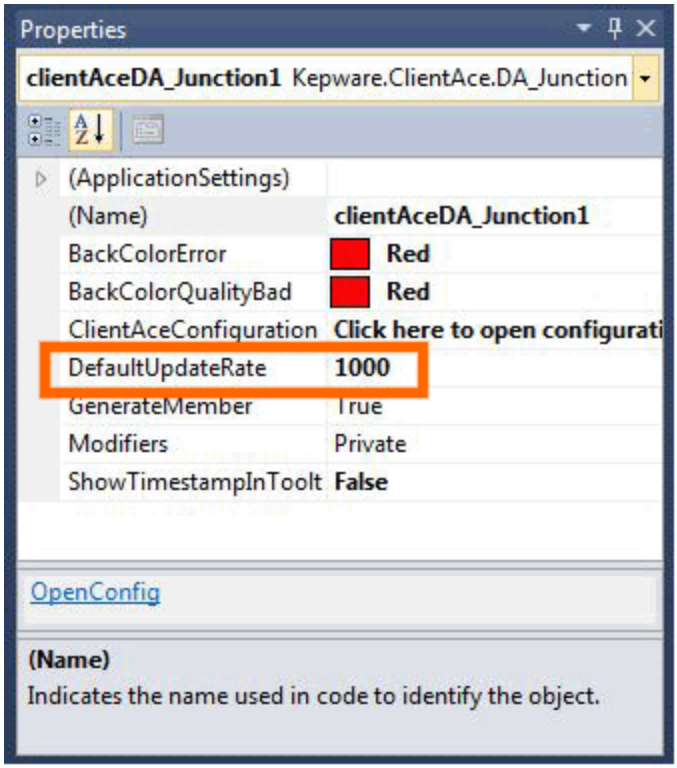
   Note: The application will use the associated VB or C# controls to read from and write to the OPC tags.

Item Update Rate

There are two update rate settings available in ClientAce: the Global Update Rate and the Item-Level Update Rate. The default setting is the Global Update Rate.

Global Update Rate

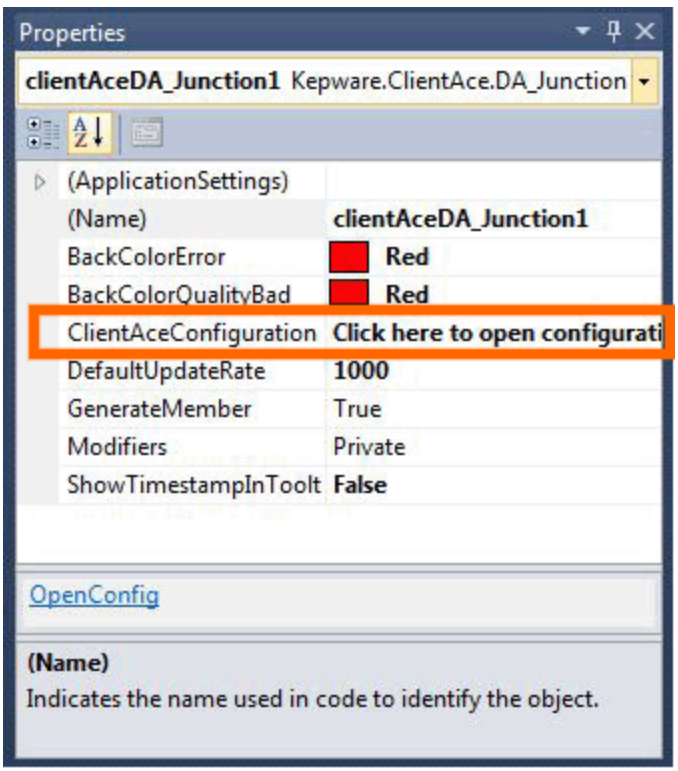
The Global Update rate specifies the default update rate for all items. The default setting is 1000 milliseconds. This setting can be modified through the **DefaultUpdateRate** property of the **DA_Junction** control.



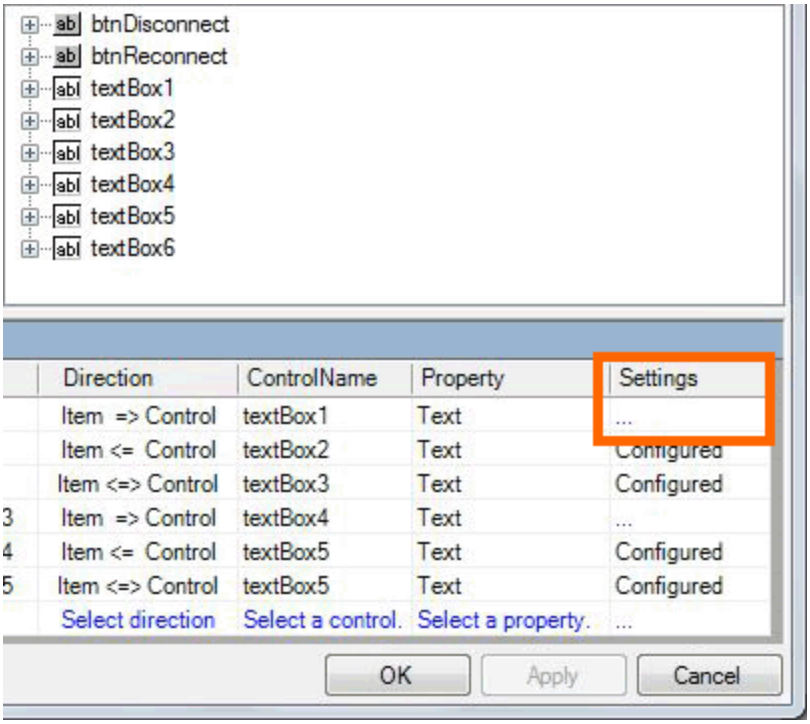
Item-Level Update Rate

Individual DA Junction items' update rates can also be modified. The change will not affect the default update rates of other controls. For more information, refer to the instructions below.

1. To start, click on the **Ellipses** button for **ClientAceConfiguration**. This will launch the **Configuration** window.

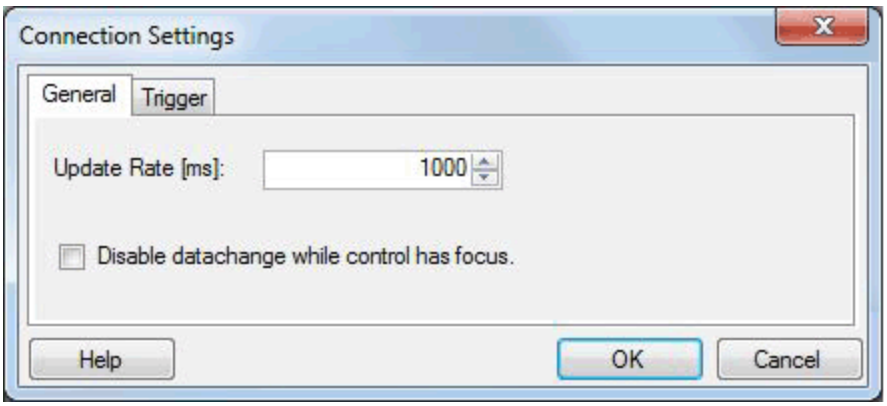


2. Next, click in the **Settings** column. Locate the item whose default rate will be changed, and then click the associated **Ellipses** button.



3. In **Connection Settings**, open the **General** tab.

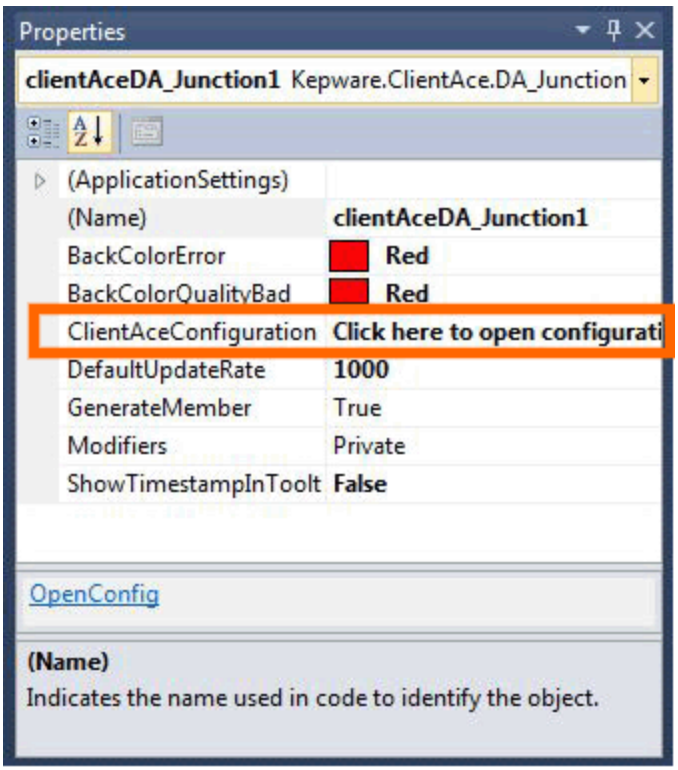
4. In **Update Rate**, change the value as desired. Then, click **OK**.



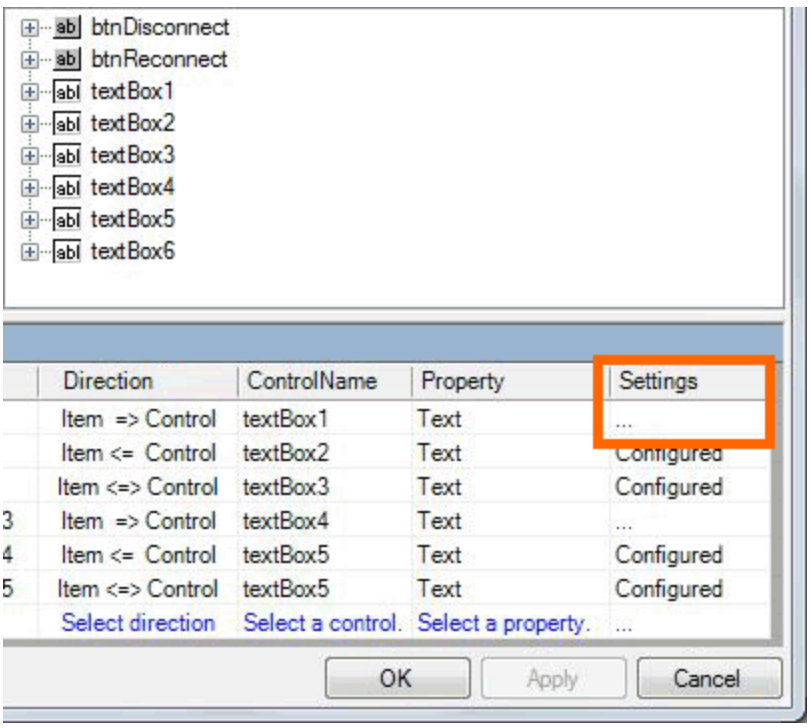
Disabling DataChange While the Control Has Focus

This parameter changes a value in the control and does not allow it to be overwritten by a change from the OPC server. For more information, refer to the instructions below.

1. To start, locate **ClientAceConfiguration**. Then, click the **Ellipses** button to launch the **Configuration** window.

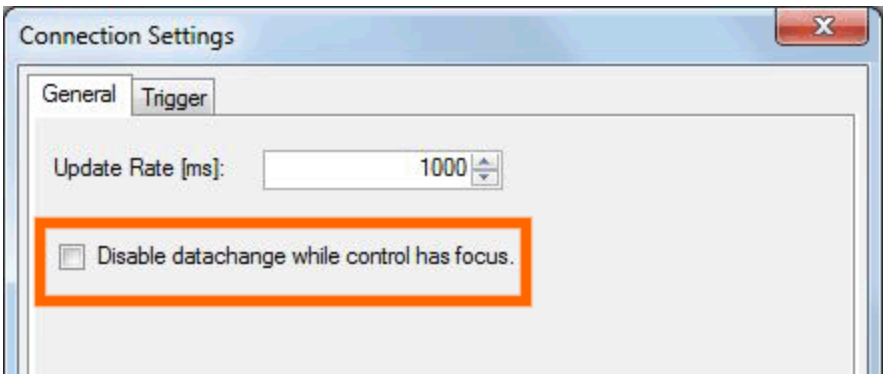


2. In the **Settings** column, locate the item whose properties will be changed. Then, click the associated **Ellipses** button.



3. In **Connection Settings**, open the **General** tab.

4. Next, check to enable **Disable datachange while control has focus**.



5. Then, click **OK**.

Note: The selected control is now set for the **Data Update Pause** when it has focus.

Additional Controls

For more information on ClientAce Browser Controls, select a link from the list below.

ItemBrowser Control Properties
ServerBrowser Control Properties

For more information on KEPServerEX Controls, refer to **KEPServerEX Controls**.

ItemBrowser Control Properties

The ItemBrowser Control lets users navigate OPC DA and OPC UA servers' address space and display items. Although these properties can only be set at the time of design, they are accessible as Read Only properties at runtime.

Property	Use	Data Type	Description
AllowMultipleServers	Input	Boolean	Indicates if multiple Servers are shown in the ItemBrowser.
BrowserWidth	Input	Integer	Indicates the width of the Tree View.
Servers	Input	OPCUrl Object	Indicates the Servers currently being used.
ShowAddServerMenuItem	Input	Boolean	Indicates if the Add Server menu items should be shown in the server browser pane when right-clicked.
ShowInternalServerBrowser	Input	Boolean	Indicates if the Internal Server Browser should be shown at Runtime.
ShowItemList	Input	Boolean	Indicates if the Item List should be shown at Runtime.
ShowItemNameAndPath	Input	Boolean	Indicates if the Item Name and Path should be shown in the Item List at Runtime.
ShowItemsInTree	Input	Boolean	Indicates if the Items should be shown in the Browser Tree List at Runtime.
ShowPropertiesInBrackets	Input	Boolean	Indicates if the Item Properties should be shown in brackets beside the Item in the Browser Tree List at Runtime.
ShowPropertiesInTree	Input	Boolean	Indicates if the Item Properties should be shown in the Browser Tree List at Runtime.
ShowPropertyList	Input	Boolean	Indicates if the Property List should be shown at Runtime.
SwitchTabPages	Input	Boolean	Indicates if the pages should switch automatically from the Item List to the Properties List when an item is selected in the Tree View List at Runtime.

AddServer Method

This method has two versions. The method below is for a string, and adds an OPC server to the Tree View of the ItemBrowser.

```
AddServer( ByVal URL as String)
```

The method below uses the URL object for OPC UA connections. This allows a UA server to be added with certificate and authentication information.

```
AddServer(byVal opcUrl As Kepware.ClientAce.BrowseControls.OpcUrl)
```

Connect Method

This method has two versions. The method below initiates a connect to the specified server in the ItemBrowser.

```
Connect( ByVal URL as String)
```

The method below is used for OPC UA connections.

```
Connect(ByVal opcUrlServer As Kepware.ClientAce.BrowseControls.OpcUrl)
```

ConnectAll Method
This method initiates a connection to all the servers currently added in the ItemBrowser.

```
ConnectAll()
```

Disconnect Method
This method has two versions. The method below initiates a disconnect to the specified server in the ItemBrowser.

```
Disconnect(ByVal URL as String)
```

The method below is used for OPC UA connections.

```
Disconnect(ByVal opcUrlServer As Kepware.ClientAce.BrowseControls.OpcUrl)
```

DisconnectAll Method
This method disconnects all servers currently connected in the ItemBrowser.

```
DisconnectAll()
```

DisconnectSelectedServer Method
This method disconnects the server currently being used. The Servernode or Childnode must be selected.

```
DisconnectSelectedServer()
```

GetSelectedItems Method
This method returns the selected items as an array of Browse Controls OPC DA items. If no item is selected, the length of the array will be 0.

```
GetSelectedItems() as Kepware.ClientAce.BrowseControls.OpcDaItem
```

ResetItemBrowser Method
This method disconnects all connected servers and clears the Tree View and lists.

```
ResetItemBrowser()
```

ItemDoubleClicked Event
This event shows that an OPC item in the browser was double-clicked.

```
ItemDoubleClicked(
ByVal Sender as Object,
ByVal item as Kepware.ClientAce.BrowseControls.OpcDaItem )
) Handles ClientAceItemBrowser1.ItemDoubleClicked
```

ItemSelected Event
This event shows that one or more OPC items are selected in the ItemBrowser.

```
ItemSelected(ByVal sender as Object, ByVal ItemCount as Integer
) Handles ClientAceItemBrowser1.ItemSelected
```

ServerAdded Event
This event shows than an OPC Server was added to the control.

```
ServerAdded(sender As Object, args As Kepware.ClientAce.BrowseControls.ItemBrowserEventArgs)
Handles ClientAceItemBrowser1.ServerAdded
```

ServerRemoved Event
This event shows than an OPC Server was removed from the control.

```
ServerRemoved(sender As Object, args As Kepware.ClientAce.BrowseControls.ItemBrowserEventArgs)
Handles ClientAceItemBrowser1.ServerRemoved
```

Parameter	Use	Functionality
URL	Input	The URL of the OPC servers. **Note:** The syntax of the URL that uniquely identifies a server must follow this format (except for OPC XML-DA): [OpcSpecification]://[Hostname]/[ServerIdentifier] OpcSpecification: Selects the OPC Specification to be used. • "opcda" for OPC Data Access 2.05A and later (COM). • **Hostname:** Name or IP Address of the machine that hosts the OPC server. For the local machine, "localhost" must be used (127.0.0.1). • **ServerIdentifier:** Identifies the OPC server on the specified host. • OPC DA (COM) – [ProgID]/[optional ClassID] **Note:** For OPC DA servers, the API will attempt to connect using the ClassID first. If the ClassID is not given (or is found to be invalid), the API will attempt to connect using the ProgID. **OPC DA Example** opcda://localhost/Kepware.KEPServerEX.V5 opcda://127.0.0.1/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397} **OPC XML-DA Example** http://127.0.0.1/Kepware/xmldaservice.asp **OPC UA Example** opc.tcp://127.0.0.1:49320

Example Code

```
[Visual Basic]
Private Sub Form3_Load( _
  ByVal sender As System.Object, _
  ByVal e As System.EventArgs _
  ) Handles MyBase.Load
 Try
  'Display the current configuration of the
  CheckBox1.Checked = ClientAceItemBrowser1.ShowAddServerMenuItem
  CheckBox2.Checked = ClientAceItemBrowser1.ShowInternalServerBrowser
  CheckBox3.Checked = ClientAceItemBrowser1.ShowItemList
  CheckBox4.Checked = ClientAceItemBrowser1.ShowItemNameAndPath
  CheckBox5.Checked = ClientAceItemBrowser1.ShowItemsInTree
  CheckBox6.Checked = ClientAceItemBrowser1.ShowPropertiesInBrackets
  CheckBox7.Checked = ClientAceItemBrowser1.ShowPropertiesInTree
  CheckBox8.Checked = ClientAceItemBrowser1.ShowPropertyList
  CheckBox9.Checked = ClientAceItemBrowser1.SwitchTabpages

  'Server to be used in the control
  ClientAceItemBrowser1.AddServer( _
  "opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}")
  ClientAceItemBrowser1.Connect( _
  "opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}")

  Catch ex As Exception
  MessageBox.Show("Received Exception: " & ex.Message)
  End Try
  End Sub
```

```
Private Sub ClientAceItemBrowser1_ItemDoubleClicked( _
ByVal sender As Object, _
ByVal item As Kepware.ClientAce.BrowseControls.OpcDaItem _
) Handles ClientAceItemBrowser1.ItemDoubleClicked
Try
'Add the item to the projects subscribed items.
mAdditems(item)
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub

Private Sub ClientAceItemBrowser1_ItemsSelected( _
ByVal sender As Object, _
ByVal itemCount As Integer _
) Handles ClientAceItemBrowser1.ItemsSelected

Try
'If more than one item is selected then add them to the projects subscribed items
If itemCount > 1 Then
mItems = ClientAceItemBrowser1.GetSelectedItems()
mAdditems(item)
End If
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub

Private Sub btnConnect_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs _
) Handles btnConnect.Click
Try
ClientAceItemBrowser1.Connect( _
"opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}")
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub

Private Sub btnDisconnect_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs _
) Handles btnDisconnect.Click
Try
ClientAceItemBrowser1.Disconnect( _
"opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}")
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub

Private Sub btnConnectAll_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs _
) Handles btnConnectAll.Click
Try
```

```
  ClientAceItemBrowser1.ConnectAll()
  Catch ex As Exception
  MessageBox.Show("Received Exception: " & ex.Message)
  End Try
  End Sub

Private Sub btnDisconnectAll_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs _
) Handles btnDisconnectAll.Click
  Try
ClientAceItemBrowser1.DisconnectAll()
Catch ex As Exception
MessageBox.Show("Received Exception: " & ex.Message)
End Try
End Sub
```

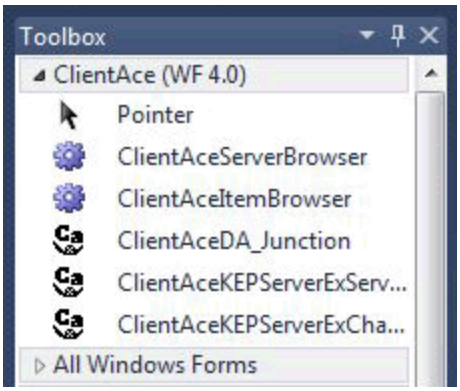
Adding an ItemBrowser Control

The ItemBrowser Control provides the functionality to browse tags in an OPC Data Access server on local or remote machines.

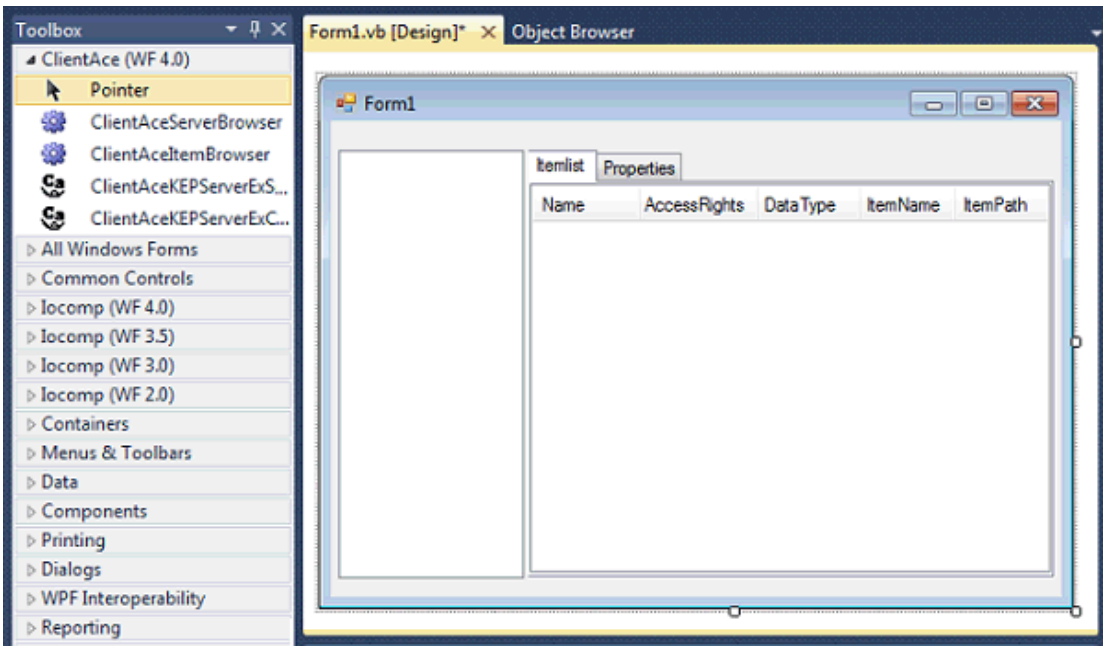
Adding the Control to the Visual Studio Project

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. To start, open a new or existing project in Visual Studio.
2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. For information on adding controls to the toolbox, refer to **Missing Controls**.

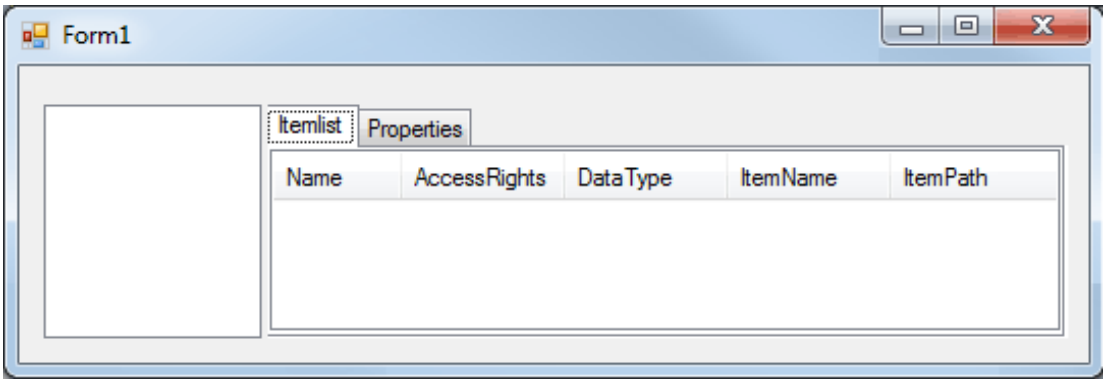


3. To add a control, drag it from the toolbox and drop it onto a form.

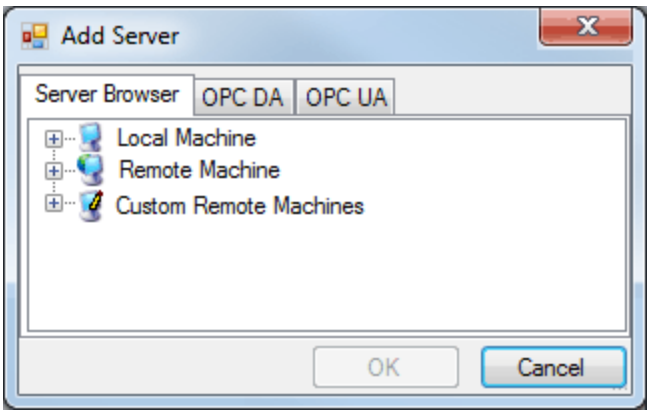


Adding a Server

1. At Runtime, the ItemBrowser Control will appear as shown in the image below. The blank left pane indicates that no servers have been added. To add a server, right-click in the left pane and then select **Add Server**.



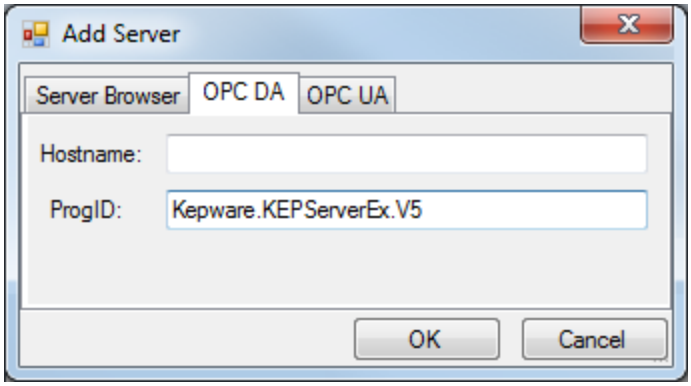
2. Next, add an OPC server using the **Server Browser**, **OPC DA**, or **OPC UA** tabs. This example will demonstrate how to add a server using the OPC DA and OPC UA tabs. For information on adding a server using the Server Browser tab, refer to **Adding a ServerBrowser Control**.



Note: When designing an application, it is best to synchronize the ItemBrowser Control with the ServerBrowser control. Do not connect to a particular server using the ServerBrowser before adding tags of a different server using the ItemBrowser Control. For more information, refer to **ServerBrowser Control**.

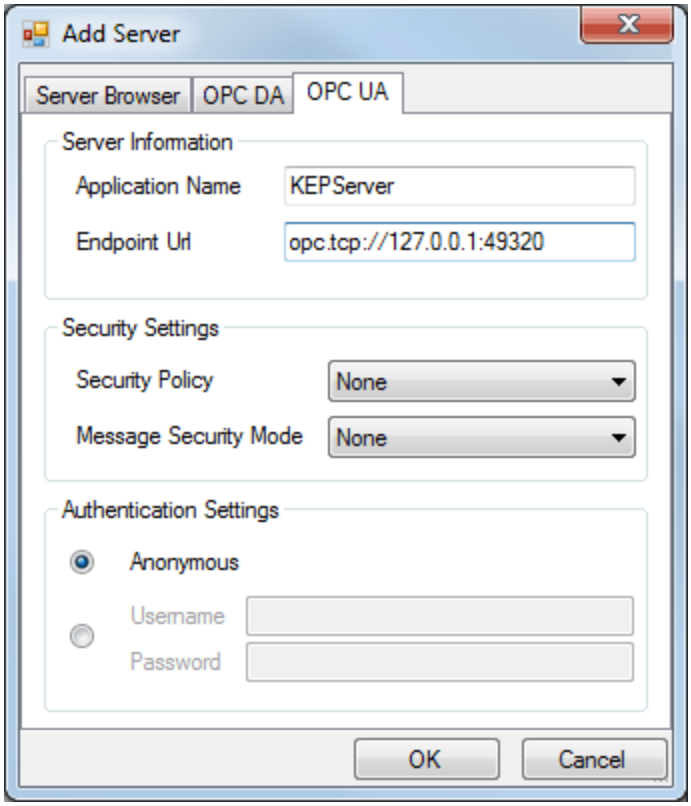
3. Select the **OPC DA** tab. Then, specify the following:

   - **Hostname:** This parameter specifies the IP Address, machine name, or localhost.
   - **ProgID:** This parameter specifies the exact ProgID of the server.

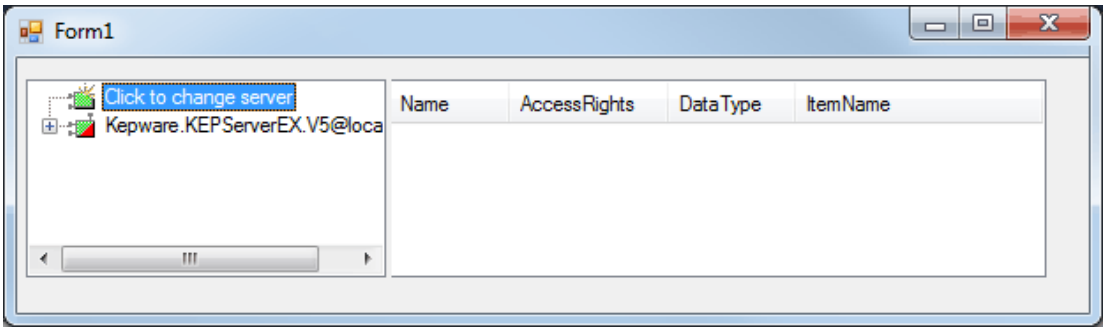


4. Next, select the **OPC UA** tab. Then, specify the following:

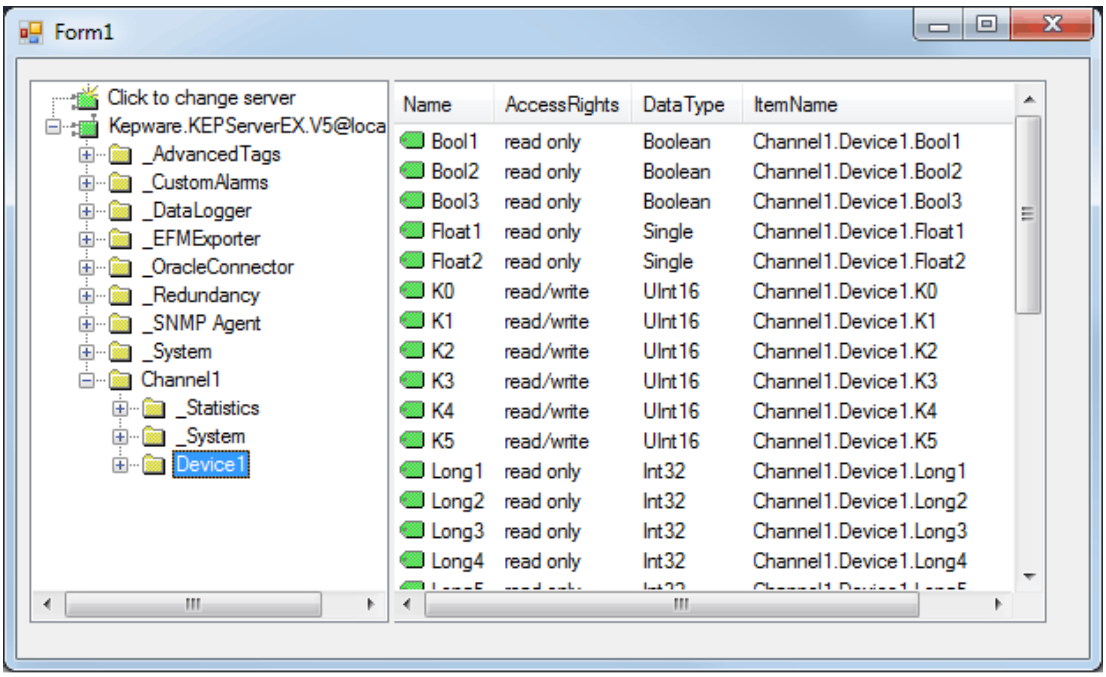
   - **Application Name:** This parameter specifies a name that will be used to identify the server.
   - **Endpoint Url:** This parameter specifies the OPC UA endpoint URL.
   - **Security Policy:** This parameter specifies the Encryption policy that is supported by the endpoint. Options include None, Basic 128 RSA 15, and Basic 256.
   - **Message Security Mode:** This parameter specifies the message security mode that is supported by the endpoint. Options include None, Sign, and Sign and Encrypt.
   - **Authentication Settings:** This parameter specifies the server's authentication settings. When user authentication is not required, select Anonymous. When user authentication is required, enter a Username and Password.



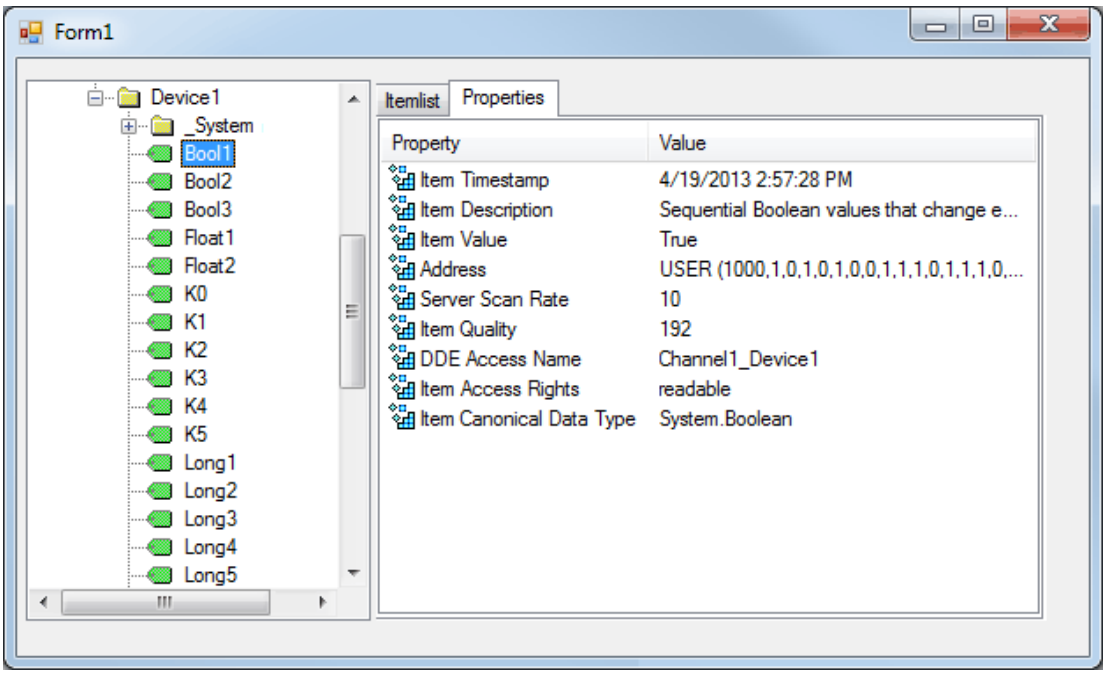
5. When finished, click **OK**. The chosen server will be placed in the left pane of the ItemBrowser window.



6. To expand the added server, locate the server name or IP Address and then click the + symbol.
7. To select the channel, click the + symbol.
8. Next, click on the tag group to display the tags in the **Itemlist** tab. In this example, the Device1 group selected from Channel1 in the server is displayed. The Device1 group's tags are displayed.



Note: The tags that are can be browsed in the ItemBrowser Control can be selected and monitored by the programming code. To view a tag's properties, click the **Properties** tab.



OpcDaItem Class

This class describes the management object for an OPC Item selected in the ItemBrowser Control.

Property	Data Type	Description
AccessRights	Object (BrowseControls.AccessRights)	The access rights of the OPC DA item.
DataType	System.Type	The description of the property. This information can be used when displaying the property in a graphical user interface (such as in a Grid Control or a

		ToolTip).
ItemName	String	The item name of the property (if the OPC Server allows properties to be read from and written to an item).
ItemPath	String	The item path of the property (if the OPC Server allows properties to be read from and written to an item).
Name	String	The display name of the OPC DA item.
ServerURL	String	The corresponding server URL.

NodeType Enumerated Values

The values shown below are the enumeration of node types for the ItemBrowser Control.

Value	Constant Name	Description
0	Server	OPC Server or Root of the Server Browse Space.
1	Branch	Branch in the address space of the OPC Server.
2	Hint	Hint that indicates how the ItemID of a Item is built.
3	Item	Item in the address space of the OPC Server.

ServerBrowser Control Properties

The ServerBrowser Control lets users search for OPC Servers on the local computer and in the network. The properties are used to set the appearance and action of the browser at Runtime. Although they can only be set at the time of design, they are accessible as Read Only properties at Runtime.

Property	Use	Data Type	Description
BrowseStatus	Input	Boolean	This is used to determine whether the Validate menu entry should be shown when a server in the browser is right-clicked.
CustomRemoteMachineCount	Input	Integer	This is used to determine how many Customer Remote Machine nodes will be displayed in the browser when they are added.
ExpandLocalMachine	Input	Boolean	This is used to determine whether the localhost node should be expanded when the browser is initialized at Runtime.
ExpandRecentlyUsedServers	Input	Boolean	Indicates if the recently-used Servers node should be initially expanded.
ExpandWebServices	Input	Boolean	Indicates if the WebServices node should be initially expanded.
OPCSpecifications	Input	ServerCategory Object	Indicates the OPC specifications to show in the browser.
RecentlyUsedServersCount	Output	Integer	Indicates the count of the recently-used Servers.
ShowCustomRemoteMachine	Input	Boolean	This is used to determine whether the custom remote machine node should be shown when the browser is initialized at Runtime.
ShowLocalMachine	Input	Boolean	This is used to determine whether the localhost node should be shown when the browser is initialized at Runtime.
ShowRecentlyUsedServers	Input	Boolean	Indicates if the recently-used Servers node should be shown.
ShowRemoteMachine	Input	Boolean	This is used to determine whether the Remote Machine network node should be shown when the browser is initialized at Runtime.
ShowWebServices	Input	Boolean	Indicates if the WebServices node should be shown.

AddServer Method

This method adds an OPC URL to the recently-used Servers node.

```
AddServer(byVal opcUrl As Kepware.ClientAce.BrowseControls.OpcUrl)
```

GetSelectedServer Method

The GetSelectedServer Method can be used to return the currently selected server's OPCUrl object or individual parts. It is used in conjunction with the ServerBrowser object's SelectionChanged and ServerDoubleclicked Events. For more information, refer to **OPCUrl Class**.

```
GetSelectedServer() As Kepware.ClientAce.BrowseControls.OpcUrl
```

ReBrowseAll Method

This method collapses all tree nodes and discards their children.

```
ReBrowseAll()
```

SelectionChanged Event

This event indicates that the selection of the OPC server in the Browse Tree has changed.

```
SelectionChanged(ByVal serverIsSelected As Boolean) Handles
ClientAceServerBrowser1.SelectionChanged
```

ServerDoubleClicked Event

This event indicates that an OPC server in the tree was double-clicked.

```
ServerDoubleClicked() Handles ClientAceServerBrowser1.ServerDoubleClicked
```

ValidateServer Method

This method validates the currently-selected server.

```
ValidateServer()
```

Example Code

```
[Visual Basic]
Private Sub CLIENTACESERVERBROWSER1_SelectionChanged(ByVal serverIsSelected As Boolean)
_
  Handles CLIENTACESERVERBROWSER1.SelectionChanged
  Dim mURL as String
  Dim mProgID as String
  Dim mOPCType as String
  Dim mCLSID as String
  Dim mHostName as String

  Try
  mURL = CLIENTACESERVERBROWSER1.GetSelectedServer.Url
  mProgID = CLIENTACESERVERBROWSER1.GetSelectedServer.ProgID
  mOPCType = CLIENTACESERVERBROWSER1.GetSelectedServer.Type.ToString
  mCLSID = CLIENTACESERVERBROWSER1.GetSelectedServer.ClsID
  mHostName = CLIENTACESERVERBROWSER1.GetSelectedServer.HostName
  mIsValid = CLIENTACESERVERBROWSER1.GetSelectedServer.IsValid

  Catch ex As Exception
  MessageBox.Show("Exception: " & ex.Message)
  End Try
  End Sub

  Private Sub CLIENTACESERVERBROWSER1_ServerDoubleClicked() _
  Handles CLIENTACESERVERBROWSER1.ServerDoubleClicked
  Dim mURL as String
  Dim mProgID as String
  Dim mOPCType as String
  Dim mCLSID as String
```

```
Dim mHostName as String
Dim mIsValid as String

Try
mURL = CLIENTACESERVERBROWSER1.GetSelectedServer.Url
mProgID = CLIENTACESERVERBROWSER1.GetSelectedServer.ProgID
mOPCType = CLIENTACESERVERBROWSER1.GetSelectedServer.Type.ToString
mCLSID = CLIENTACESERVERBROWSER1.GetSelectedServer.ClsID
mHostName = CLIENTACESERVERBROWSER1.GetSelectedServer.HostName
mIsValid = CLIENTACESERVERBROWSER1.GetSelectedServer.IsValid
Catch ex As Exception
MessageBox.Show("Exception: " & ex.Message)
End Try
End Sub
```

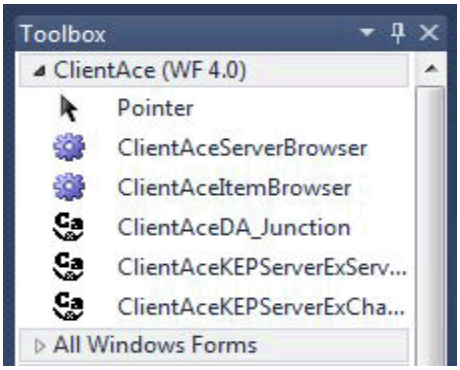
Adding a ServerBrowser Control

The ServerBrowser Control provides the functionality to browse OPC Data Access servers on local and remote machines.

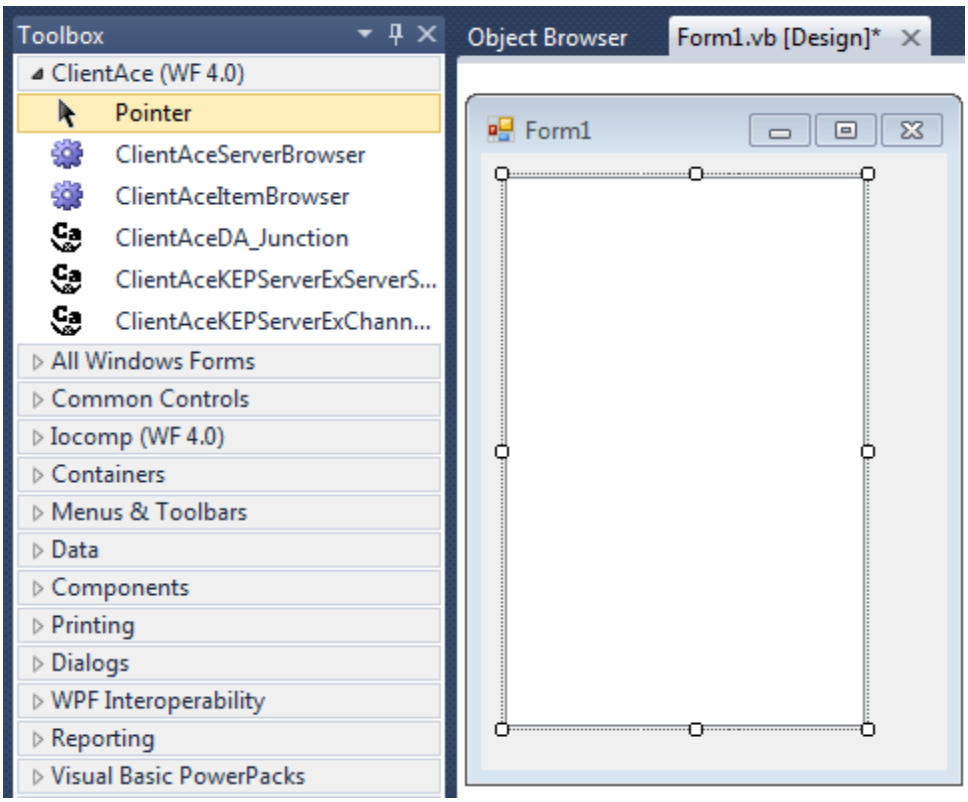
Adding the Control to the Visual Studio Project

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. To start, open a new or existing project in Visual Studio.
2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. For information on adding controls to the toolbox, refer to **Missing Controls**.

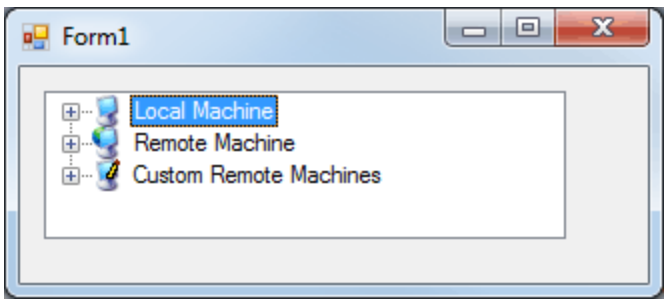


3. To add a control, drag it from the toolbox and drop it onto a form.



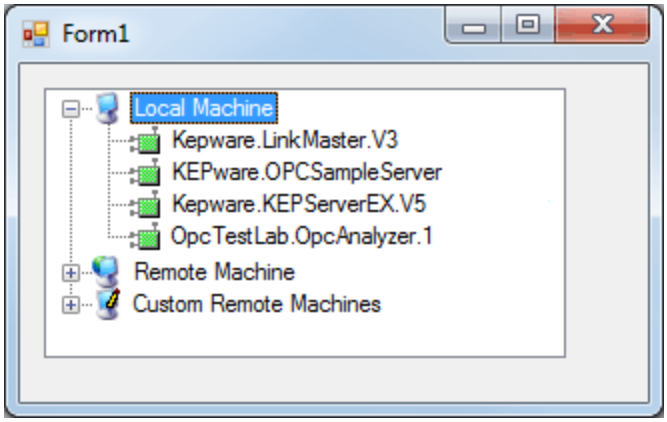
The ServerBrowser Control at Runtime

At Runtime, the ServerBrowser Control will appear as shown in the image below.



Local Machine

To expand the Local Machine and display the servers, click on the + symbol. To select a server, simply click on it. For more information on using the ClientAce API to connect to the server, refer to **ClientAce .NET API Assembly**.



Remote Machine

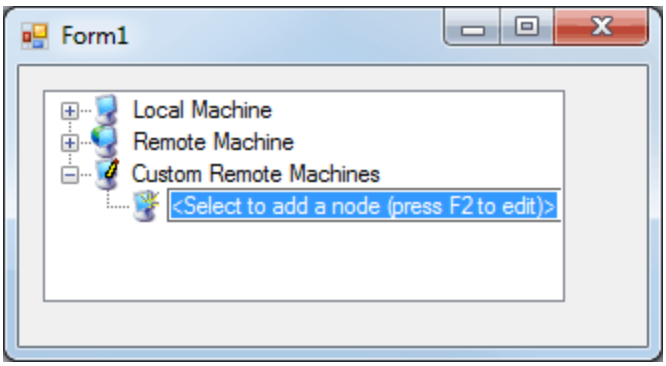
To expand the Remote Machine and display the nodes/machines on the network, click on the + symbol. To display all Enumerated OPC DA servers and all OPC UA servers that are registered with a Local UA Discovery Server, click on the + symbol. The Remote Machine's DCOM settings must be configured properly for users to access its servers. To select a server, simply click on it.

Note: For more information on using the ClientAce API to connect to the server, refer to **ClientAce .NET API Assembly**.

Adding a Custom Remote Machine

Custom Remote Machines are used to custom define links to remote machines (using either the IP Address or machine name of the PC that will be browsed). For information on defining a custom link to a remote machine, refer to the instructions below.

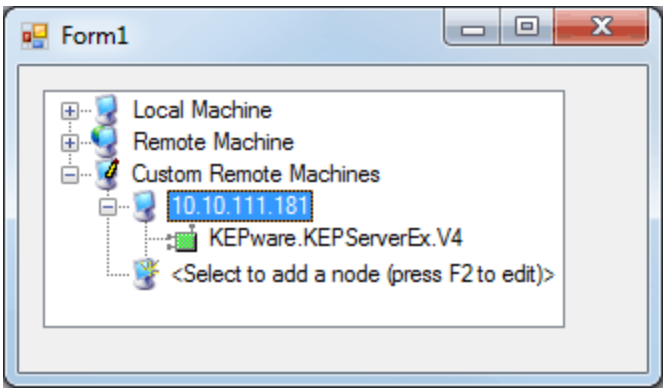
1. To start, locate **Custom Remote Machines** and then click the + symbol.
2. Click **<Select to add a node>** and then press **F2**.



3. Next, type the IP Address or the machine name of the remote PC that will be browsed.
4. Press **Enter**.

   Note: This will create a link that points to the remote machine. To display the servers on the remote machine, click on the + symbol next to the remote machine IP Address or name.

5. Next, click on a server to highlight it.



Note: In this example, the remote machine 10.10.111.181 has been defined as a custom link.

Important: Once a Custom Remote Machine has been created, the link will be saved by the application. The next time that the application is opened, the Custom Remote Machine will be available and accessible; however, it is only associated with the application that it was created for originally. For example, when a new application is created, the Custom Remote Machines created for other applications/projects will not be available for browsing. A new Custom Remote Machine link must be created for that new application/project.

OPCType Enumerated Values

The values shown below are the enumeration for the OPC specification types.

Value	Constant Name	Description
0	NOTDEFINED	No type defined. This is the default state.
1	XMLDA	OPC XML Data Access.
2	DA	OPC Data Access.
3	AE	OPC Alarm and Events.
4	DX	OPC Data Exchange.
5	HDA	OPC Historical Data Access.
6	UA	OPC Unified Architecture.

OPCUrl Class

This class describes the management object for the URL of an OPC Server selected in the ServerBrowser Control and ItemBrowser Control.

Property	Data Type	Description
ClsID	String (BrowseControls.OPCType)	The registered Class ID of the selected OPC Server.
HostName	String	The name of the host machine where the selected OPC Server is located. For a local server connection, this is called the "localhost."
IsValid	Boolean	Reports whether or not the selected server is a valid OPC Server.
ProgID	String	The Program ID for the selected COM OPC Server.
Type	Object*	The OPC Specification Type (such as DA) for an OPC DA Server.*
URL	String	The complete OPC server's URL takes the following form: OPC DA: opcda://[Hostname, e.g. localhost]/[ProgID]/[ClsID] OPC XML-DA: http://[Hostname, e.g. localhost]/[location of service file] OPC UA: opc.tcp://[Hostname, e.g. localhost]:[Port]

*For more information, refer to **OPCType Enumerated Values**.

KEPServerEX Controls

ChannelSetting Control

OPC DA

Although these properties can only be set at the time of design, they are accessible as Read Only properties at Runtime.

Property	Data Type	Description
ChannelName	String	The Channel Name of the channel in the server to which the control is connected.
NodeName	String	The location of the server to which the control is connected. This is called the "localhost" in a local connection, and the IP Address of the Host Name for a remote connection.
ProgID	String	The Program ID of the server to which the Channel Settings Control is connected.

See Also: **Adding a ChannelSetting Control**

ServerState Control

OPC DA

Although the properties can only be set at the time of design, they are accessible as Read Only properties at Runtime.

Property	Data Type	Description
NodeName	String	The location of the server to which the control is connected. This is called the "localhost" for a local connection, and the IP Address of the Host Name for a remote connection.
ProgID	String	The Program ID of the server to which the Channel Settings Control is connected.

See Also: **Adding a ServerState Control**

Adding a ChannelSetting Control

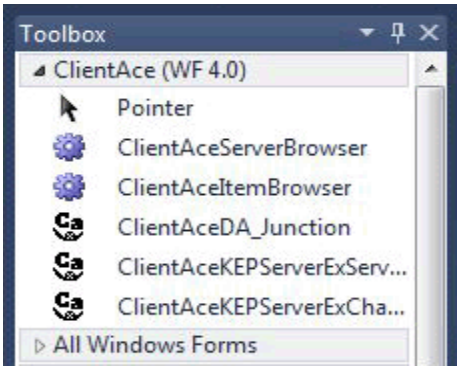
The ChannelSetting Control provides the functionality to view and make certain changes to the properties of a channel in an OPC server provided by Kepware Technologies.

Note: If multiple KEPServerEX OPC servers are installed on the local machine, the ChannelSetting Control will retrieve the channel properties of the server that was installed most recently.

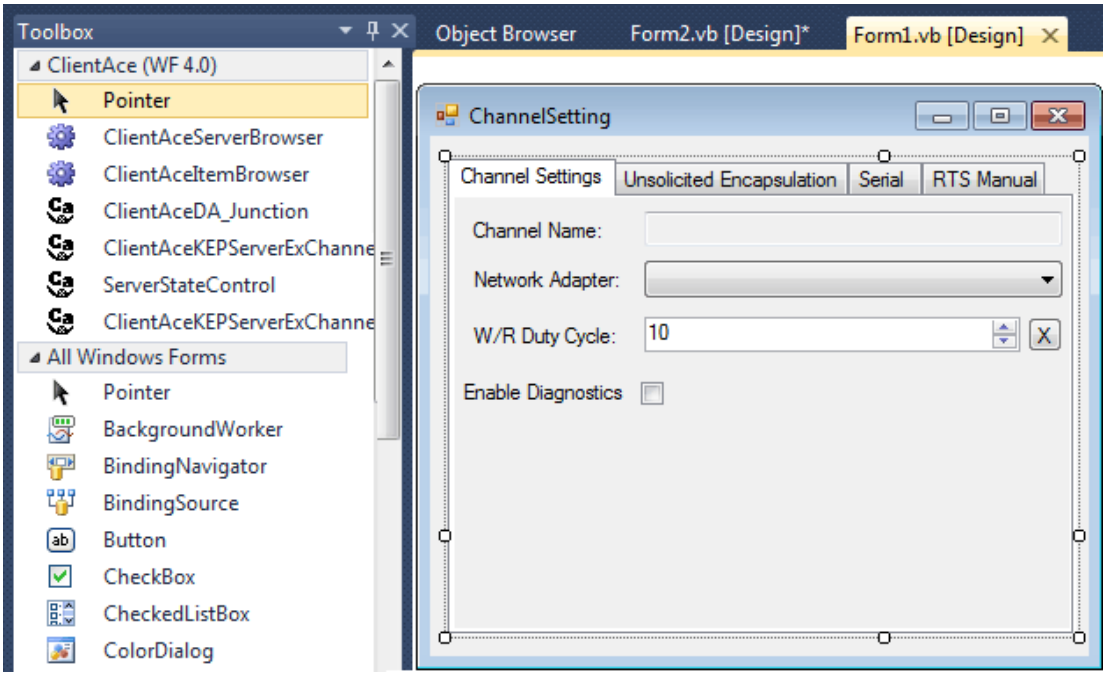
Adding the Control to the Visual Studio Project

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. To start, open a new or existing project (solution) in Visual Studio.
2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. In Visual Studio, the Toolbox should include the controls shown below. For more information on adding controls to the toolbox, refer to **Missing Controls**.

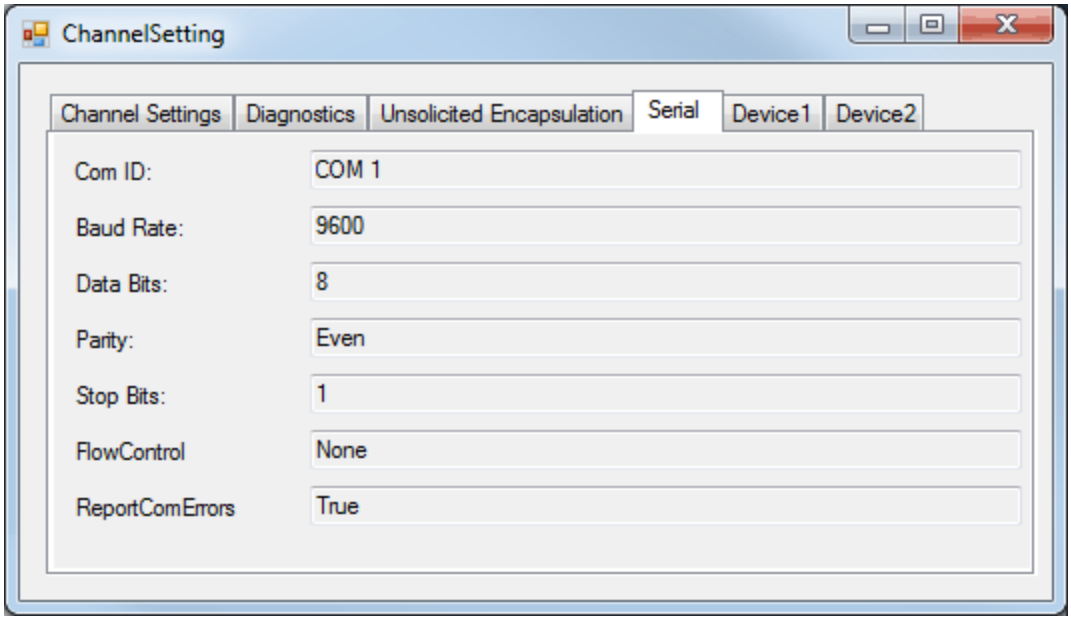


3. To add a control, drag it from the toolbox and drop it onto a form.



The ChannelSetting Control at Runtime

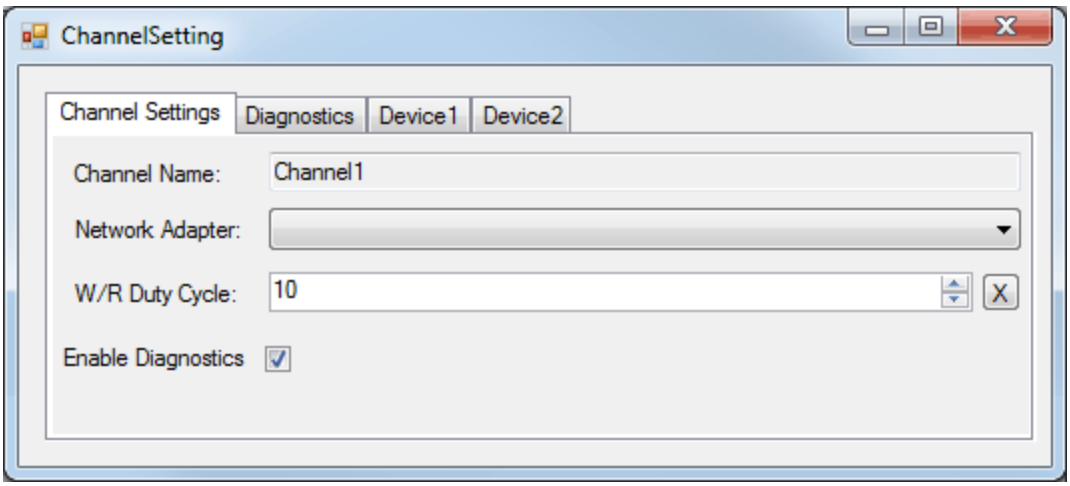
The ChannelSetting Control will display different tabs depending on the type of channel to which it is linked. The image below shows the options available to serial channels.



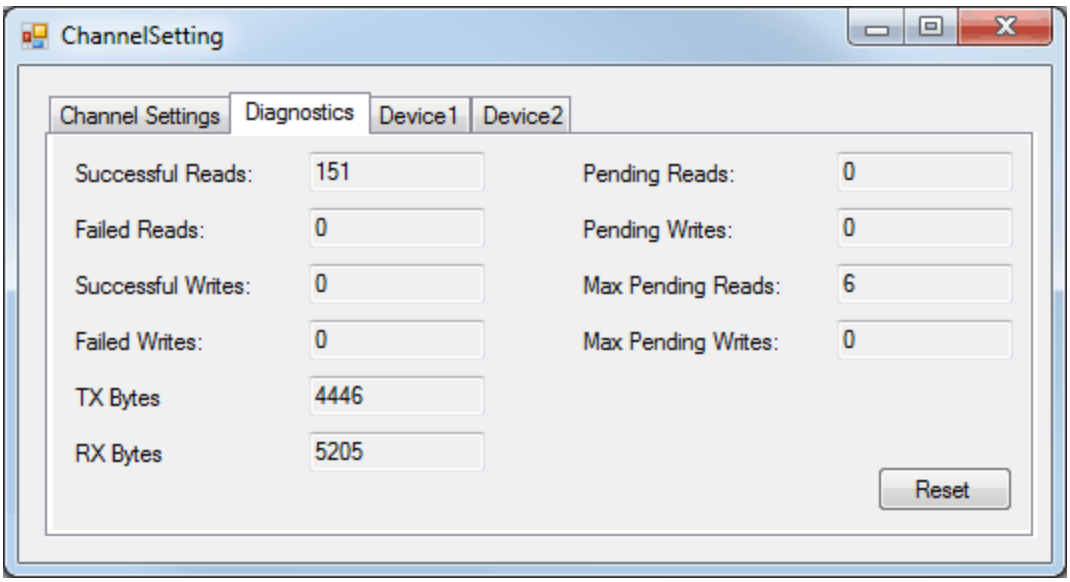
Linking a ChannelSetting Control to a Specific Channel

1. To start, right-click on the **ChannelSetting** Control and then select **Properties**.
2. In **Channel Name**, enter "Channel1" (because that node name is present in the sample KEPServerEX OPC project).

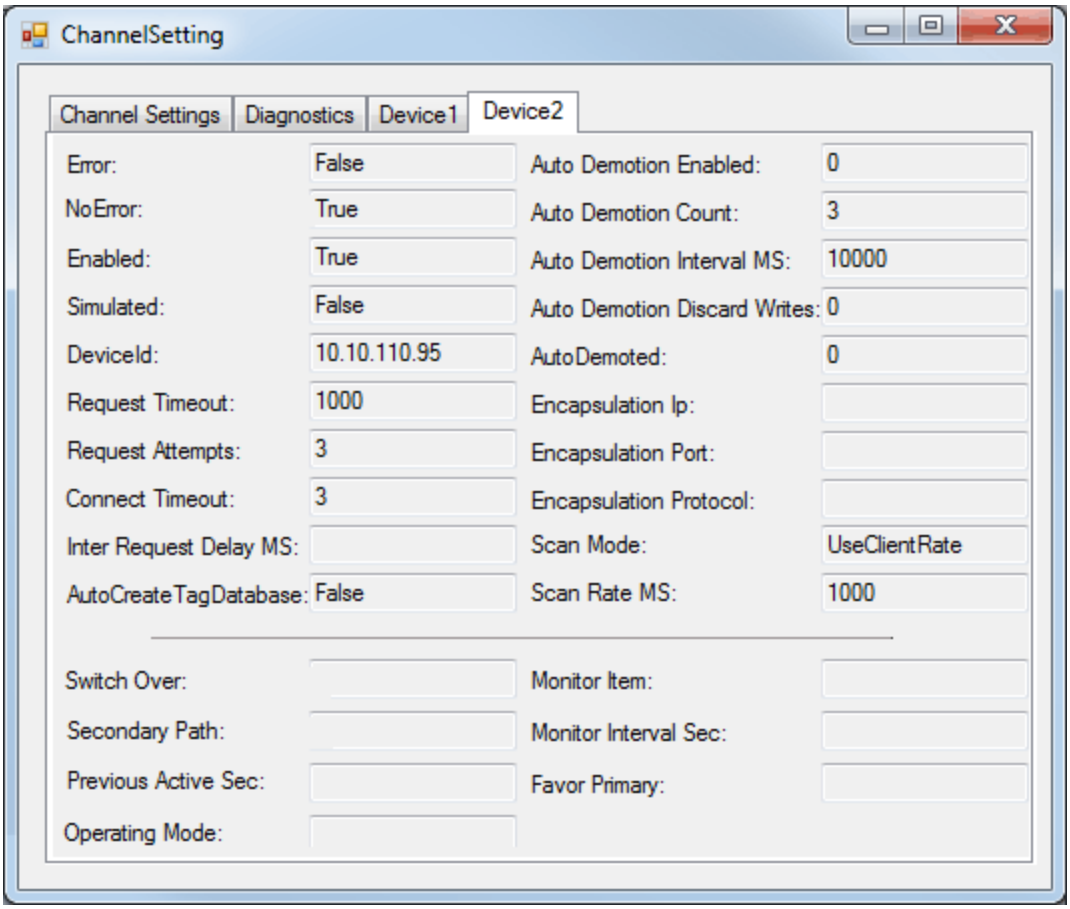
   Note: If the channel uses a network adapter, it will be listed in the **Network Adapter** parameter. Both the Network Adapter and W/R Duty Cycle field can be modified as needed.
3. Next, specify the **Enable Channel Diagnostics** setting.



Note: When enabled, diagnostics information will be displayed in a separate **Diagnostics** tab.



Note: The **Device1** and **Device2** tabs display the properties of the two devices configured under the channel. The window will display a tab for each device that is configured. The Device Properties cannot be modified in this window even though they are displayed.



Adding a ServerState Control

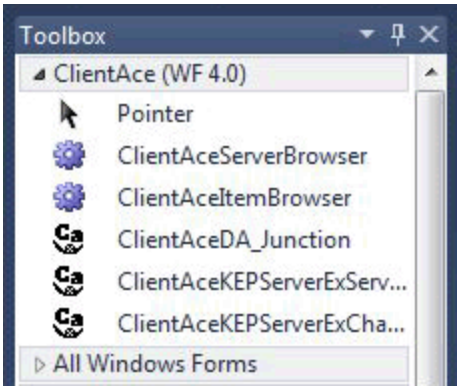
The ServerState Control provides the functionality to view and make certain changes to the properties of the project in an OPC server provided by Kepware Technologies.

Note: If multiple KEPServerEX OPC servers are installed on the local machine, the ServerState Control will retrieve the project properties of the server that was installed most recently.

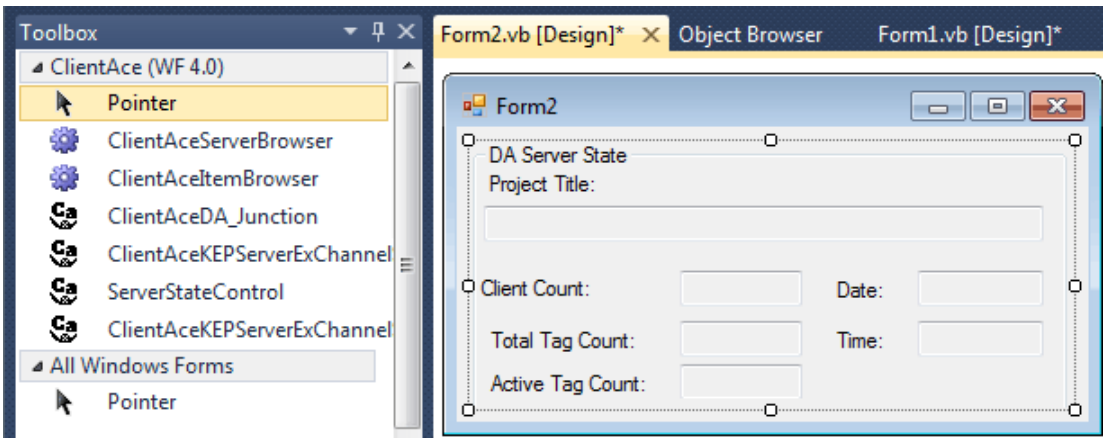
Adding the Control to the Visual Studio Project

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. To start, open a new or existing project in Visual Studio.
2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. In Visual Studio, the toolbox should include the controls shown below. For information on adding controls to the Toolbox, refer to **Missing Controls**.

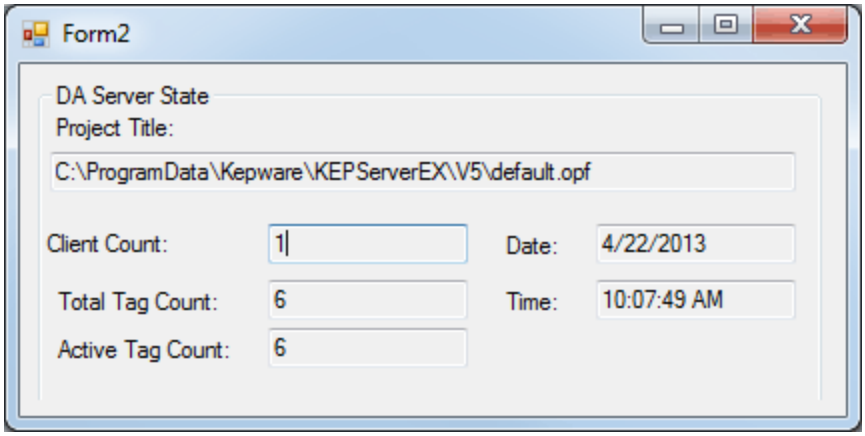


3. To add a control, drag it from the toolbox and drop it onto a form.



The ServerState Control at Runtime

At Runtime, the ServerState Control will appear as shown below.



Note: Initially, the tag count displayed in the **Total Tag Count** and **Active Tag Count** fields is 6. This accounts for the six state properties that are displayed: Client Count, Total Tag Count, Active Tag Count, Date, Time, and Project Name.

Data Types Description

OPC Type	Visual Studio Type	Description
Boolean	Boolean	Single bit. The following value range depends on implementation: True = 1 and -1 False = 0
Word	UShort or UInt16	Unsigned 16 bit value (2 bytes). The value range is 0 to 65535.
Short	Int16	Signed 16 bit value (2 bytes). The value range is -32768 to 32767.
DWord	UInteger or Uint32	Unsigned 32 bit value (4 bytes). The value range is 0 to 4294967295.
Long	Integer or Int32	Signed 32 bit value (4 bytes). The value range is -2,147,483,648 to 2,147,483,647.
Float	Single	32 bit floating point value (4 bytes). The value range depends on implementation: For negative values: -3.4028235E+38 to -1.401298E-45 .* For positive values: 1.401298E-45 to 3.4028235E+38.*
Double	Double	64 bit floating point value (8 bytes). The value range depends on implementation: For negative values: -1.79769313486231570E+308 to -4.94065645841246544E-324.* For positive values: 4.94065645841246544E-324 to 1.79769313486231570E+308.*
String	String	Typically null terminated, null padded, or blank padded ASCII string. 0 to 2 billion Unicode characters in Visual Studio.
Char	SByte	1 byte. The value range is -128 to 127.
Date	Date or DateTime Long or Int64 ULong or UInt64 Char or Wide Char Decimal	8 bytes.The value range is 0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999. 8 bytes. The value range is -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18).* 8 bytes. The value range is 0 through 18,446,744,073,709,551,615 (1.8...E+19). 2 bytes. The value range is 2 to 65535.** 16 bytes. The value range is as follows: 0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point.* 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal.

*The smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28).
**Unicode characters. Some drivers support Unicode strings as individual data (but not wide characters).

Applying ClientAce

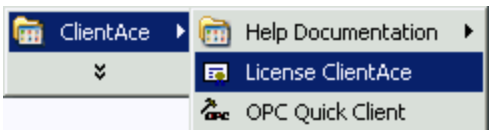
For more information on applying ClientAce, select a link from the list below.

Licensing ClientAce

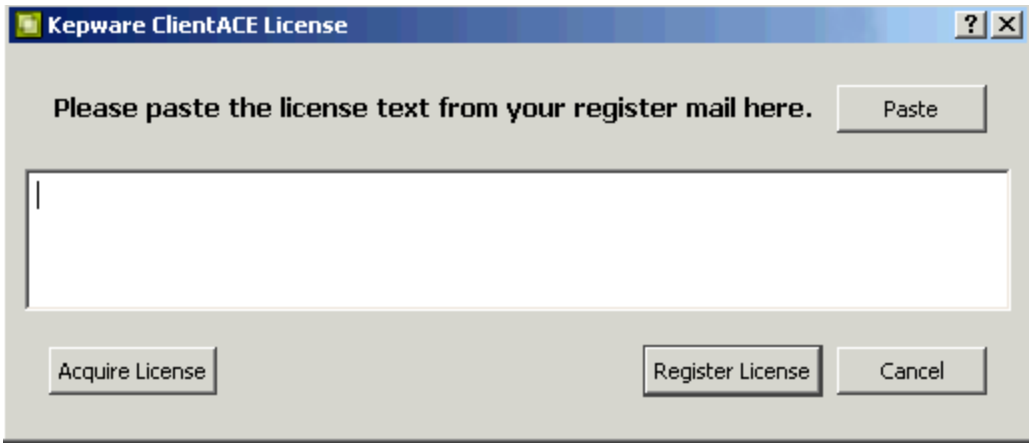
Unless ClientAce is licensed (and all Runtime applications built with the ClientAce .NET controls have been signed), the applications will run in Demo Mode for one hour. After the demo period expires, another demonstration period can be started by restarting the application. After ClientAce is licensed and the Runtime applications built with ClientAce .NET controls are signed, the applications will run in unlimited Runtime operation. For information on licensing ClientAce, refer to the instructions below.

Note: For all licensing questions, contact Kepware Technologies at sales@kepware.com or +1-207-775-1660 ext. 211.

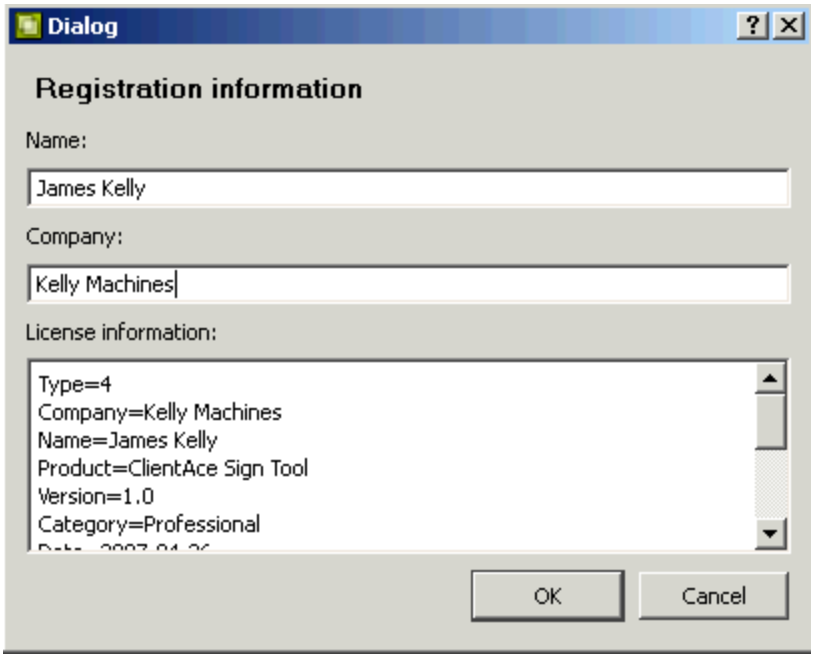
1. To start, click **Start** | **Programs** | **Kepware Products**.
2. Then, click **ClientAce** | **License ClientAce**.



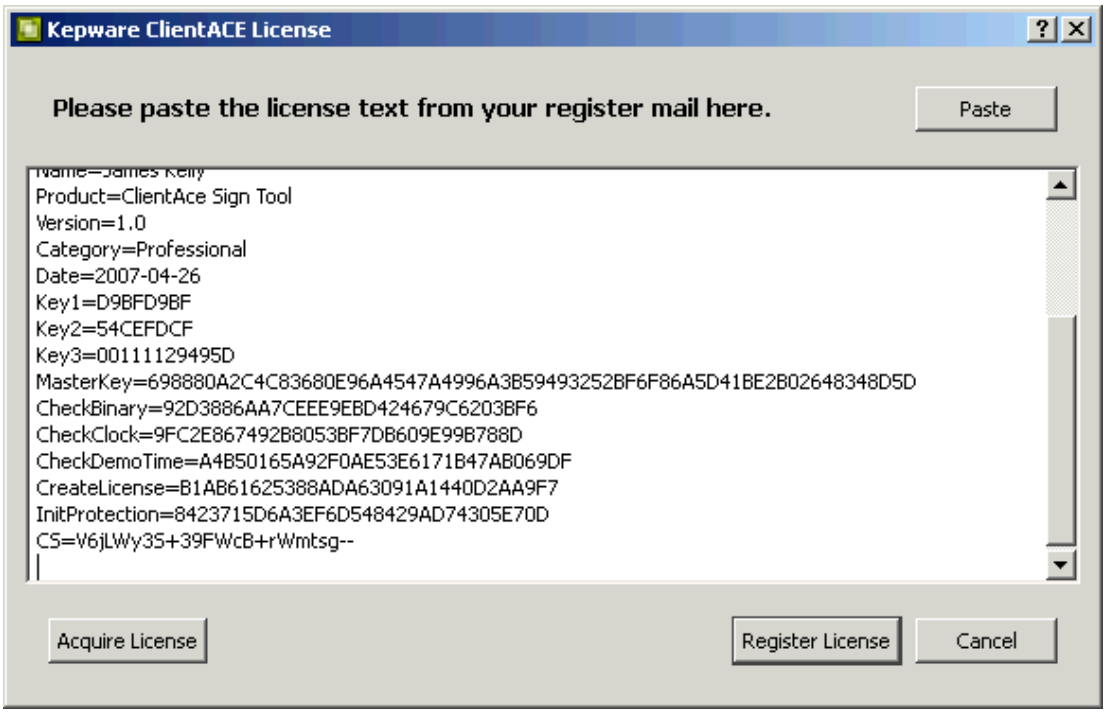
3. In **Kepware ClientACE License**, click **Acquire License**.



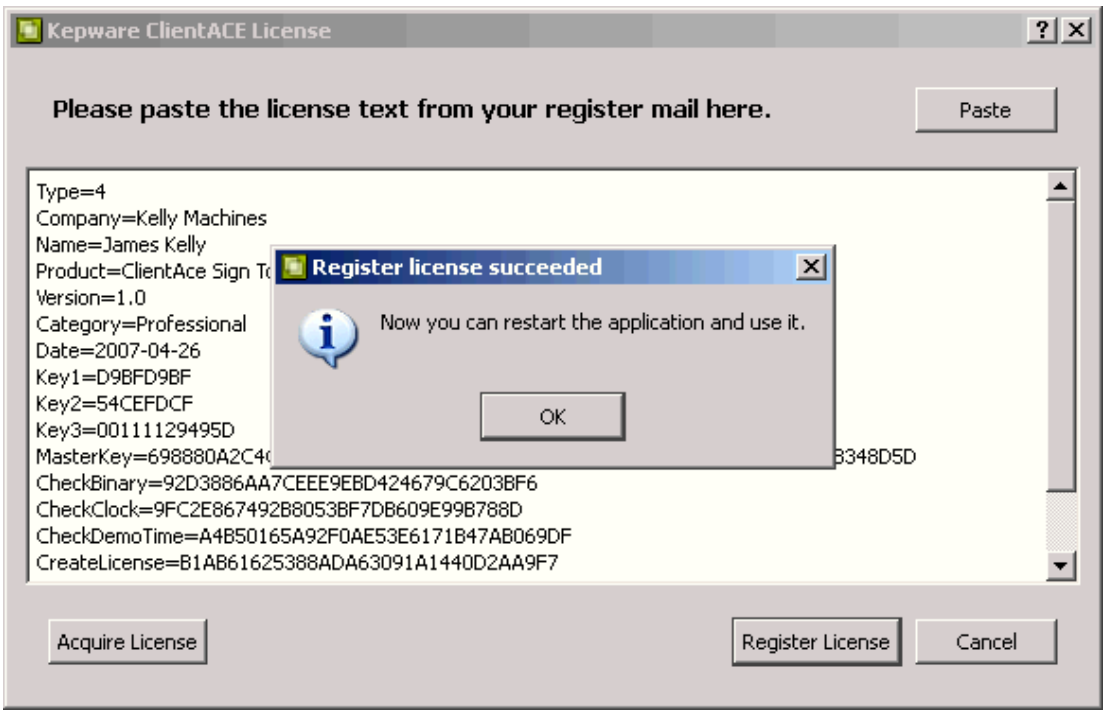
4. In **Registration Information**, specify the **Name** and **Company** fields. As the information is entered, the **License Information** field will be populated with the licensing information needed by Kepware Technologies.



5. Once finished, click **OK**. This will invoke an email message window from the email client application. To send the message to Kepware Technologies, click **Send**.
6. Kepware Technologies will then send an email reply containing the licensing code. Copy the code into the **Kepware ClientACE License** dialog.



7. Then, click **Register License**.
8. Once the confirmation message is displayed, click **OK**.



Note: ClientAce is now licensed. The custom client applications that have been built may now be signed. For more information, refer to **Signing Your Client Application**.

Upgrading ClientAce

When upgrading a ClientAce project to a newer version, users should do the following:

1. Clean the project's Bin folders by removing all the files that have been created using the old .dll files. This is necessary because the project must be created using the new .dll files, and some of the old files may not be removed when the Solution Explorer is cleaned.
2. In the project's references, remove the current ClientAce .dll files that are referenced. Then, replace them with the new .dll files.
3. Recompile and test the new project to ensure that it works using the new .dll files.

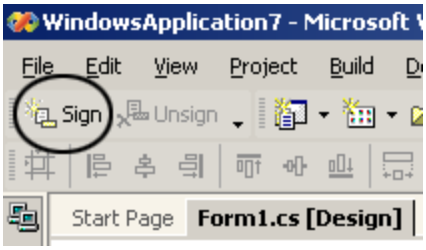
Signing a Client Application

Applications created using a ClientAce .NET controls must be signed before they can run for unlimited Runtime operation. If the application is not signed, it will run in demo mode.

Note: ClientAce must be licensed from Kepware Technologies before applications can be signed. For more information, refer to **Licensing ClientAce**.

Signing the Custom Client Application Using the Visual Studio Sign Add-in

1. To start, open the project that needs to be signed.
2. Next, click the **Sign** icon in the toolbar. This tags the project's executable file to be signed whenever the project is built.

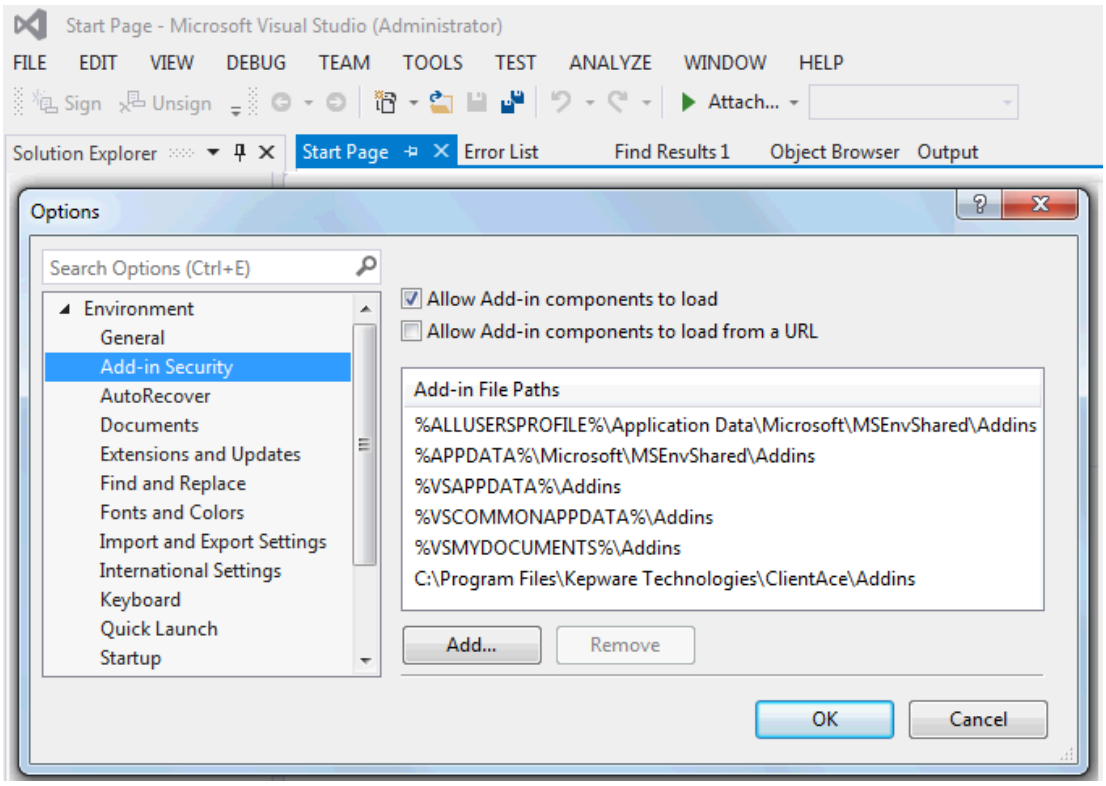


Notes:

1. The license file (*.lic) is saved in the same folder as the executable file.
2. Signing the ClientAce applications will add the following two lines to the application's Post Build events:

   C:\Program Files\Kepware Technologies\ClientAce\Sign\sign.exe" "$(TargetPath)" "$(TargetName).lic
   C:\Program Files\Kepware Technologies\ClientAce\Sign\sign.exe" "$(TargetDir) $(TargetName).vshost.exe" "$(TargetName).vshost.lic

3. The install adds the ClientAce Add-in path to the IDE Add-in Security Option in Tools | Options | Environment | Add-in Security | Add-in File Paths.

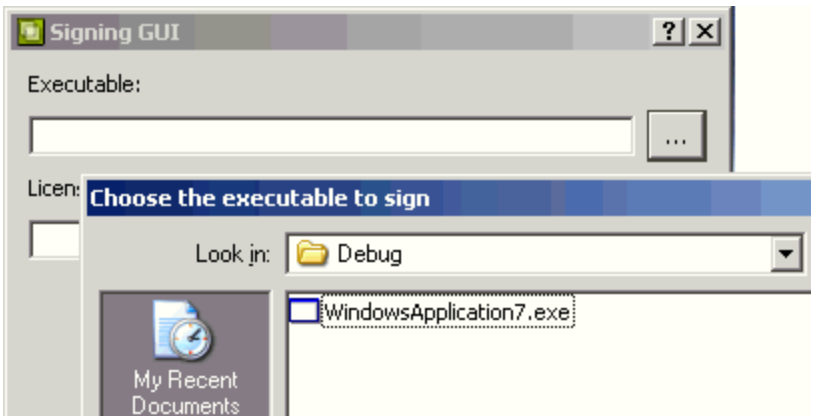


Manually Signing the Custom Client Application

When the application is signed manually, the steps must be repeated every time the project is built to sign the application. For more information, refer to the instructions below.

1. To start, click **Start** | **Programs** | **Kepware Products**.
2. Then, click **ClientAce** | **Sign Executable**.

3. In **Signing GUI**, click the **Ellipses** button to browse for the application's executable file.



4. When the executable file is chosen, the signed license code will be displayed in the **License File** field. The license file (*.lic) will be saved in the same folder as the executable file.
5. Once finished, click **OK** to save and exit.

Note: The license file (*.lic) will be saved to the same folder that is chosen for the build output path in **Project Compile Preferences**.

- In Visual Studio 2003 and Visual Studio 2005, the default output path is in *bin\Debug* in the project folder.
- In Visual Studio 2008, the default output path is in *bin\Release*.

As a result of this change, Visual Studio 2008 users run in Demo Mode (and receive the Demo Mode popup) when testing a project in Debug Mode that has been signed. To change this behavior, change the output path to *\bin\Debug*.

Creating a Setup Project

Creating a Set-up Project to Deploy ClientAce Applications in Licensed Mode

Users must add the ClientAce .LIC file to the set-up project and set the project dependencies so that a current .LIC file is generated every time the set-up project is built. For information on creating a Visual Studio set-up project so that a ClientAce application is deployed in Licensed Mode, refer to the instructions below.

These steps need to be completed once for each ClientAce project.

1. To start, sign the ClientAce EXE project *(See "Signing a Client Application" on page 117)*.
2. Build the ClientAce project to generate an .LIC file.
3. Add the set-up project to the Visual Studio solution that contains the EXE project. In the Solution Explorer, right-click on the solution and click **Add | New Project...**. Alternatively, if the solution or any project is selected in the Solution Explorer, click **File | Add... | New Project** in the Visual Studio menu.
4. In Add New Project, select **Other Project Types | Setup and Deployment | Setup Project**.
5. Create a name for the set-up project. For example "Project_Setup".
6. Once the set-up project appears in the Solution Explorer, right-click on the setup project and select **Add | Project Output...**.
7. In Add Project Output Group, locate the Project field and select the ClientAce EXE project.
8. In the list box in the middle of the dialog,verify at least "Primary output" is selected.
9. Click **OK**.
10. Right-click on the setup project and select **Add | File...**.

11. In Add Files, browse to the "bin\Release\" folder of the ClientAce EXE project (or other folder according to the build configuration to be deployed. Select the .LIC file.
12. In the Project Explorer, right-click on the solution and choose **Project Dependencies...**.
13. Locate the Dependencies tab and select the setup project.
14. In the list in the middle of the dialog, check the **ClientAce EXE project** checkbox.
15. Click **OK**.
16. Right-click on the solution and select **Configuration Manager...**.
17. In the list at the upper-left corner, select the configuration to be built. This usually matches the folder from step 11.
18. Locate the setup project in the table in the middle of the dialog, and check the **Build** checkbox.
19. Click **OK**.
20. When users build and deploy the setup project, the *.LIC file is included with the *.EXE. This runs the application in Licensed Mode.

Note: The instructions above have been tested with a Windows Application project in Visual Studio 2010 and 2013. The steps should be similar for other types of projects.

Deploying a Client Application

Depending on the version of Visual Studio used, it may be necessary to download and install the Visual Studio Installer Extension Project from Microsoft.

For information on a specific version of Visual Studio and the .NET Assemblies, select a link from the list below:

- **Visual Studio 2003 and Visual Studio 2005 (.NET 2.0.0.x Assemblies)**
- **Visual Studio 2008 (.NET 3.5.0.x Assemblies)**
- **Visual Studio 2010, 2012, and 2013 (.NET 4.0.2.x Assemblies)**

Visual Studio 2003 and Visual Studio 2005 (.NET 2.0.0.x Assemblies)

Depending on the ClientAce features being used by the application, one or more of the following files may be required for the application to run properly:

Name	Version
Kepware.ClientAce.Base.dll	2.0.0.x
Kepware.ClientAce.BrowseControls.dll	2.0.0.x
Kepware.ClientAce.Da_Junction.dll	2.0.0.x
Kepware.ClientAce.KEPServerExControls.dll	2.0.0.x
Kepware.ClientAce.OpcClient.dll	2.0.0.x

YourCustomClientAceApplication.exe
YourCustomClientAceApplication.lic

These files will be located in the output build directory created by Visual Studio for the project. When deploying the client application created using ClientAce and the .NET 2.0.0.x Assemblies, these files must be installed in the same location as the custom client executable files.

.NET Framework Requirements

.NET Framework 2.0 must be installed on the PC on which the client will deploy custom client applications created using ClientAce and the .NET 2.0.0.x Assemblies. If the client application utilizes functionality from a .NET Framework version that is higher than the .NET 2.0 Framework, then that version must also be installed. To check if .NET Framework is installed, follow the instructions below.

1. Click **Start** on the Windows desktop, and then select the **Control Panel**.
2. Next, double-click **Add or Remove Programs**.

3. Next, scroll through the list of applications. If Microsoft .NET Framework 2.0 is listed, the version required by ClientAce is already installed and does not need to be installed again.
4. To obtain versions of the .NET Framework, click **Start** on the Windows desktop and then select **Windows Update**.

Note: The actual ClientAce install does not need to be installed on the destination computer for the custom ClientAce application to work.

See Also: System and Application Requirements

Visual Studio 2008 (.NET 3.5.0.x Assemblies)

Depending on the ClientAce features being used by the application, one or more of the following files may be required for the application to run properly:

Name	Version
Kepware.ClientAce.BrowseControls.dll	3.5.0.x
Kepware.ClientAce.Da_Junction.dll	3.5.0.x
Kepware.ClientAce.KEPServerExControls.dll	3.5.0.x
Kepware.ClientAce.OpcClient.dll	3.5.0.x

YourCustomClientAceApplication.exe
YourCustomClientAceApplication.lic

These files will be located in the project's output build directory that was created by Visual Studio. When deploying the client application created using ClientAce and the .NET 3.5.0.x Assemblies, these files must be installed in the same location as the custom client executable files.

.NET Framework Requirements

.NET Framework 3.5 Service Pack 1 must be installed on the PC on which the client deploys the custom client applications created using ClientAce and the .NET 3.5.0.x Assemblies. If the client application utilizes functionality from a .NET Framework version that is higher than the .NET 3.5 Framework, then that version must also be installed. To check if the .NET Framework is installed, follow the instructions below.

1. Click **Start** on the Windows desktop, and then select the **Control Panel**.
2. Next, double-click **Add or Remove Programs**.
3. Next, scroll through the list of applications. If Microsoft .NET Framework 3.5 SP1 is listed, the version required by ClientAce is already installed and does not need to be installed again.
4. To obtain versions of the .NET Framework, click **Start** on the Windows desktop and then select **Windows Update**.

Note: The actual ClientAce install does not need to be installed on the destination computer for the custom ClientAce application to work.

See Also: System and Application Requirements

Visual Studio 2010, 2012, and 2013 (.NET 4.0.2.x Assemblies)

Depending on the ClientAce features being used by the application, one or more of the following files may be required for the application to run properly:

Name	Version
Kepware.ClientAce.BrowseControls.dll	4.0.2.x
Kepware.ClientAce.Da_Junction.dll	4.0.2.x
Kepware.ClientAce.KEPServerExControls.dll	4.0.2.x
Kepware.ClientAce.OpcClient.dll	4.0.2.x

YourCustomClientAceApplication.exe
YourCustomClientAceApplication.lic

These files will be located in the project's output build directory that was created by Visual Studio. When deploying the client application created using ClientAce and the .NET 4.0.2.x Assemblies, these files must be installed in the same location as the custom client executable files.

.NET Framework Requirements

.NET Framework 4.0 must be installed on the PC on which the client deploys the custom client applications created using ClientAce and the .NET 4.0.2.x Assemblies. If the client application utilizes functionality from a .NET Framework version that is higher than the .NET 4.0 Framework, then that version must also be installed. To check if the .NET Framework is installed, follow the instructions below.

1. Click **Start** on the Windows desktop, and then select the **Control Panel**.
2. Next, double-click **Add or Remove Programs**.
3. Next, scroll through the list of applications. If Microsoft .NET Framework 4.0 is listed, the version required by ClientAce is already installed and does not need to be installed again.
4. To obtain versions of the .NET Framework, click **Start** on the Windows desktop and then select **Windows Update**.

Note: The actual ClientAce install does not need to be installed on the destination computer for the custom ClientAce application to work.

See Also: **System and Application Requirements**

Troubleshooting

For more information on a common troubleshooting problem, select a link from the list below.

ASP .NET Development Incompatibility

ClientAce cannot be used to develop ASP .NET applications. If ASP .NET OPC clients must be developed, please contact Kepware Technical Support.

CoInitializeSecurity

The ClientAce application must set its security credentials such that an OPC server has the privilege to send OnDataChange/OnServerShutDown notifications to the client. To set the security credentials, a ClientAce application must set the security level using CoInitializeSecurity during the application's initialization.

Example Code

The Visual Basic and C# examples below show how to call CoInitializeSecurity in the ClientAce application.

```
[Visual Basic]
' .Net library for Interoperability
ImportsSystem.Runtime.InteropServices
' declaring the enum for the CoInitializeSecurity call
PublicEnum RpcImpLevel
E_Default = 0
E_Anonymous = 1
E_Identify = 2
E_Impersonate = 3
E_Delegate = 4
EndEnum
PublicEnum EoAuthnCap
E_None = &H0
E_MutualAuth = &H1
E_StaticCloaking = &H20
E_DynamicCloaking = &H40
E_AnyAuthority = &H80
E_MakeFullSIC = &H100
E_Default = &H800
E_SecureRefs = &H2
E_AccessControl = &H4
E_AppID = &H8
```

```
E_Dynamic = &H10
E_RequireFullSIC = &H200
E_AutoImpersonate = &H400
E_NoCustomMarshal = &H2000
E_DisableAAA = &H1000
End Enum

Public Enum >RpcAuthnLevel
E_Default = 0
E_None = 1
E_Connect = 2
E_Call = 3
E_Pkt = 4
E_PktIntegrity = 5
E_PktPrivacy = 6
EndEnum

'end of enums declared for the CoInitializeSecurity call

Public Class Form1

Inherits System.Windows.Forms.Form

' declare the CoInitializeSecurity signature within the class where it
' should be called (must be called before launching form

Declare Function CoInitializeSecurity Lib "ole32.dll" (ByVal pVoid As IntPtr, _
ByVal cAuthSvc As Integer, ByVal asAuthSvcByVal As IntPtr, _
ByVal pReserved1 As IntPtr, ByVal dwAuthnLevel As Integer, ByVal dwImpLevel As Integer, _
ByVal pAuthList As IntPtr, ByVal dwCapabilities As Integer, ByVal pReserved3 As IntPtr) As Integer

#Region " Windows Form Designer generated code "

Public Sub New()
MyBase.New()

' good place to call CoInitializeSecurity
CoInitializeSecurity(IntPtr.Zero, -1, IntPtr.Zero, _
IntPtr.Zero, RpcAuthnLevel.E_None, _
RpcImpLevel.E_Impersonate, IntPtr.Zero, EoAuthnCap.E_None, IntPtr.Zero)

'This call is required by the Windows Form Designer.

InitializeComponent()

'Add any initialization after the InitializeComponent() call
End Sub
```

```
C#]

// .net library required for interoperability

usingSystem.Runtime.InteropServices;

// ******Enums required for CoInitializeSecurity call through C#.....//

publicenum RpcImpLevel
{ Default = 0,
Anonymous = 1,
Identify = 2,
Impersonate = 3,
Delegate = 4
}

publicenum EoAuthnCap
{ None = 0x00,
MutualAuth = 0x01,
StaticCloaking= 0x20,
DynamicCloaking= 0x40,
AnyAuthority= 0x80,
MakeFullSIC= 0x100,
Default= 0x800,
SecureRefs= 0x02,
AccessControl= 0x04,
AppID= 0x08,
Dynamic= 0x10,
RequireFullSIC= 0x200,
AutoImpersonate= 0x400,
NoCustomMarshal= 0x2000,
DisableAAA= 0x1000
}

publicenum RpcAuthnLevel

{ Default = 0,
None = 1,
Connect = 2,
Call = 3,
Pkt = 4,
PktIntegrity = 5,
PktPrivacy = 6
}

/*****************end of enum declarations for CoInitializeSecurity call******/
```

```
namespace CSharpTestClient
{

publicclass Form1 : System.Windows.Forms.Form
{ // Import the CoInitializeSecurity call from
[DllImport("ole32.dll", CharSet = CharSet.Auto)]

public static extern int CoInitializeSecurity( IntPtr pVoid, int
cAuthSvc,IntPtrasAuthSvc, IntPtr pReserved1, RpcAuthnLevel level, RpcImpLevel impers,IntPtr
pAuthList, EoAuthnCap dwCapabilities, IntPtr
pReserved3 );

private Kepware.ClientAce.DA_Junction.ClientAceDA_Junction clientAceDA_Junction1;

private System.Windows.Forms.TextBox textBox1;

public Form1()

{

InitializeComponent();

}

///

///The main entry point for the application.

///

[STAThread]

static void Main()

{

// call the CoInitializeSecurity right before Launching the Application

CoInitializeSecurity( IntPtr.Zero, -1, IntPtr.Zero,
IntPtr.Zero,RpcAuthnLevel.None ,
RpcImpLevel.Impersonate,IntPtr.Zero, EoAuthnCap.None, IntPtr.Zero );

Application.Run(new Form1());

}

}

}
```

```
[C#]
// .net library required for interoperability
usingSystem.Runtime.InteropServices;
```

```
// ******Enums required for CoInitializeSecurity call through C#...../
publicenum RpcImpLevel
 { Default = 0, Anonymous = 1,
  Identify = 2, Impersonate = 3,
  Delegate = 4 }
publicenum EoAuthnCap
 { None = 0x00,
  MutualAuth = 0x01,
  StaticCloaking= 0x20,
  DynamicCloaking= 0x40,
  AnyAuthority= 0x80,
  MakeFullSIC= 0x100,
  Default= 0x800,
  SecureRefs= 0x02,
  AccessControl= 0x04,
  AppID= 0x08,
  Dynamic= 0x10,
  RequireFullSIC= 0x200,
  AutoImpersonate= 0x400,
  NoCustomMarshal= 0x2000,
  DisableAAA= 0x1000 }
publicenum RpcAuthnLevel
  { Default = 0, None = 1,
  Connect = 2, Call = 3,
  Pkt = 4, PktIntegrity = 5,
  PktPrivacy = 6 }
/*****************end of enum declarations for CoInitializeSecurity call******/
(Continued)
namespace CSharpTestClient
{
publicclass Form1 : System.Windows.Forms.Form
  { // Import the CoInitializeSecurity call from
  [DllImport("ole32.dll", CharSet = CharSet.Auto)]
  public static extern int CoInitializeSecurity( IntPtr pVoid, int
cAuthSvc,IntPtrasAuthSvc, IntPtr pReserved1, RpcAuthnLevel level, RpcImpLevel impers,IntPtr
pAuthList, EoAuthnCap dwCapabilities, IntPtr
  pReserved3 );
privateKepware.ClientAce.DA_Junction.ClientAceDA_Junction clientAceDA_Junction1;
  private System.Windows.Forms.TextBox textBox1;
  public Form1()
  {
  InitializeComponent();
}
  /// <summary>
  ///The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main()
  {
// call the CoInitializeSecurity right before Launching the Application
  CoInitializeSecurity( IntPtr.Zero, -1, IntPtr.Zero,
  IntPtr.Zero,RpcAuthnLevel.None ,
RpcImpLevel.Impersonate,IntPtr.Zero, EoAuthnCap.None, IntPtr.Zero );
  Application.Run(new Form1());
  }
  }
}
```

Converting Visual Studio 2008 to Visual Studio 2010

Using Visual Studio 2008 Examples with Visual Studio 2010

Visual Studio 2008 examples may be used with Visual Studio 2010 after they have been converted to Visual Studio 2010 solutions. To do so, utilize the Visual Studio Conversion Wizard. Afterward, the examples may be compiled and run.

Installing Visual Studio 2010 when Visual Studio 2008 and ClientAce are Currently Installed

For information on installing Visual Studio 2010 when Visual Studio 2008 and ClientAce are already installed, refer to the instructions below.

1. To start, install **Visual Studio 2010**. Then, run the program.
2. In **Choose Default Environment Settings**, select the desired environment.
3. Once finished, click **Start Visual Studio**. Then, close Visual Studio.
4. Next, run the ClientAce setup and select **Modify**. Then, continue through the installation.

   Note: This procedure is recommended because the Sign Toolbar and ClientAce Toolbox will not successfully migrate from Visual Studio 2008. As a result, both the Sign Toolbar and the ClientAce Toolbox added to Visual Studio 2010 will be invalid.

Repairing the Invalid Sign Toolbar and ClientAce Toolbox Added by Migrate Settings

Users whose install of Visual Studio 2010 migrated settings from Visual Studio 2008 can use the following procedure to repair the invalid Sign Toolbar and ClientAce Toolbox.

1. Run the **ClientAce** setup and select **Modify**.
2. Then, continue through the installation.

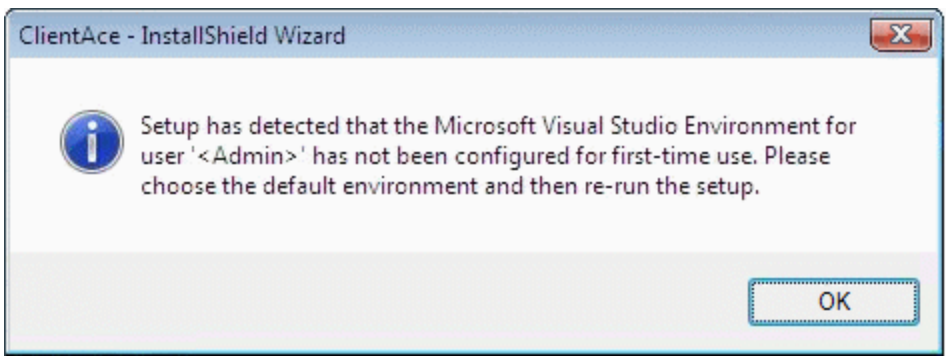
Manually Removing the Sign Toolbar and ClientAce Toolbox Added by Migrate Settings

Users who do not want to use ClientAce with Visual Studio 2010 (or whose migration of the Visual Studio 2008 settings added an invalid Sign Toolbar and ClientAce Toolbox) can use the following procedure to manually remove the Sign Toolbar and ClientAce Toolbox.

1. To start, open **Visual Studio 2010**.
2. Locate the **Toolbox** window. Then, right-click and select **Show All**.
3. Next, right-click on **ClientAce Tab** and select **Delete Tab**.
4. Then, click **View** | **Toolbars** | **Customize**.
5. Locate the **Kepware Sign Bar** and then select **Delete**.

Microsoft Visual Studio Environment Configuration

While running the ClientAce setup, users may be presented with the following message:



At this point, the specified user must run Microsoft Visual Studio and finish setting up the default Visual Studio environment. Once completed, the ClientAce setup may continue.

Note: The ClientAce setup cannot add toolbars or toolbox items until the Visual Studio environment has been configured for the current user.

Missing Controls

The following controls are typically added to the system's Visual Studio Environment automatically during the ClientAce installation process. If the Toolbox does not have any of the ClientAce controls, it is possible that the controls were unchecked during the ClientAce installation process.

Required ClientAce Controls

- DA_Junction
- ServerBrowser
- ItemBrowser

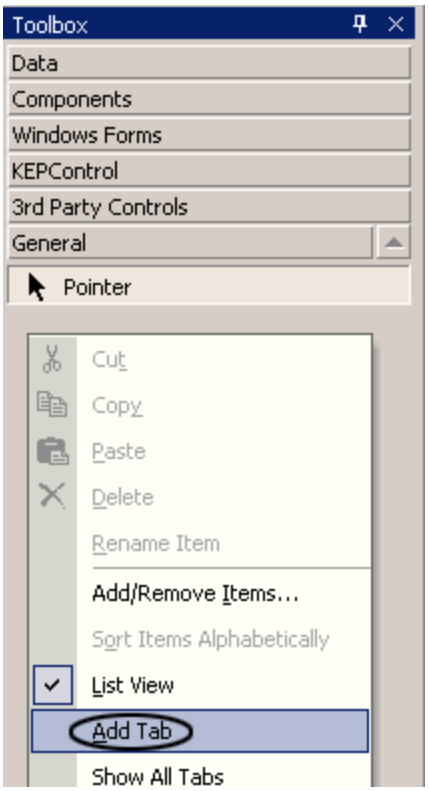
Optional KEPServerEX Controls

- ChannelSetting
- ServerState

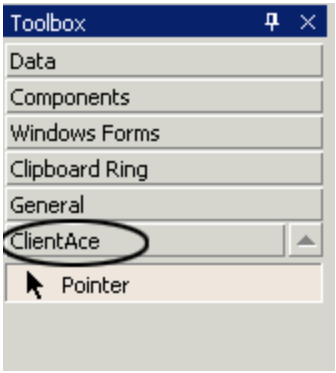
Adding ClientAce Controls to the Visual Studio Environment

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. To start, open a new C# or Visual Basic project using the Visual Studio .Net application.
2. Then, right-click anywhere on the Toolbox window and select **Add Tab**.

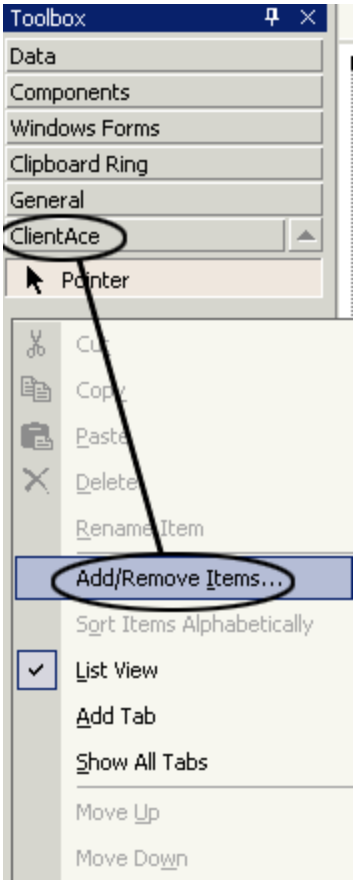


3. In the empty box, enter "ClientAce". This will create a ClientAce tab.

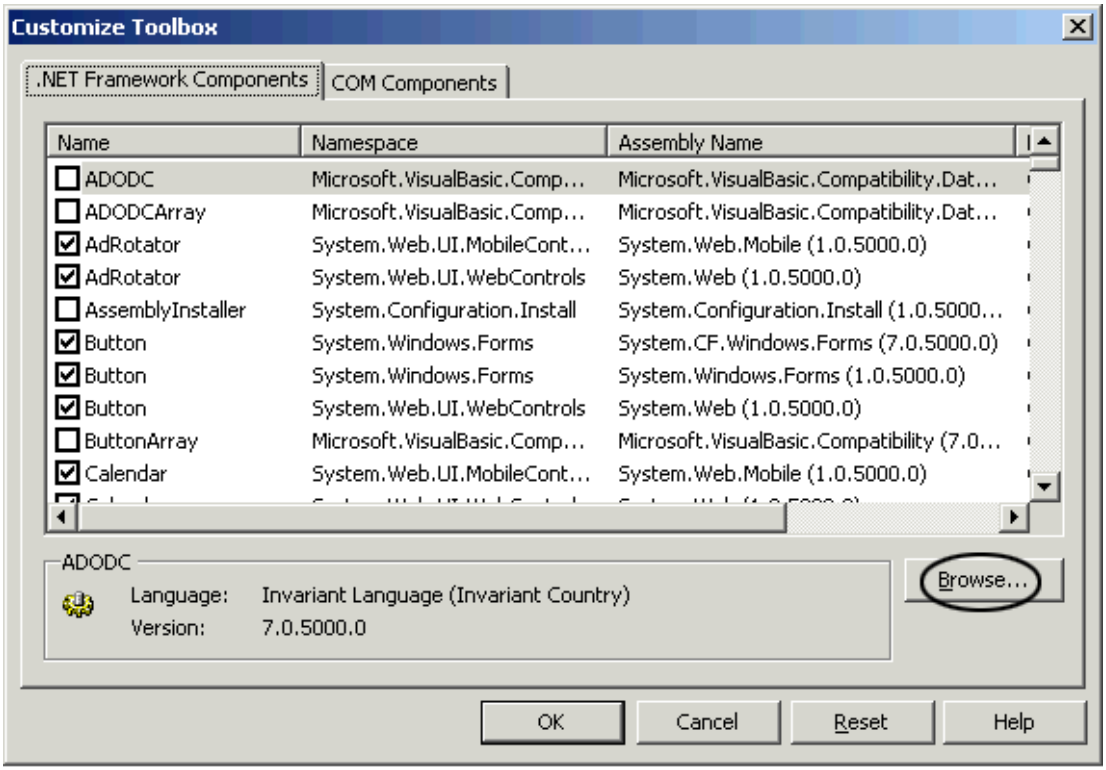


4. Next, right-click anywhere on the ClientAce tab and select **Add/Remove Items**.

   Note: If using Visual Studio 2005, select **Choose Items**.

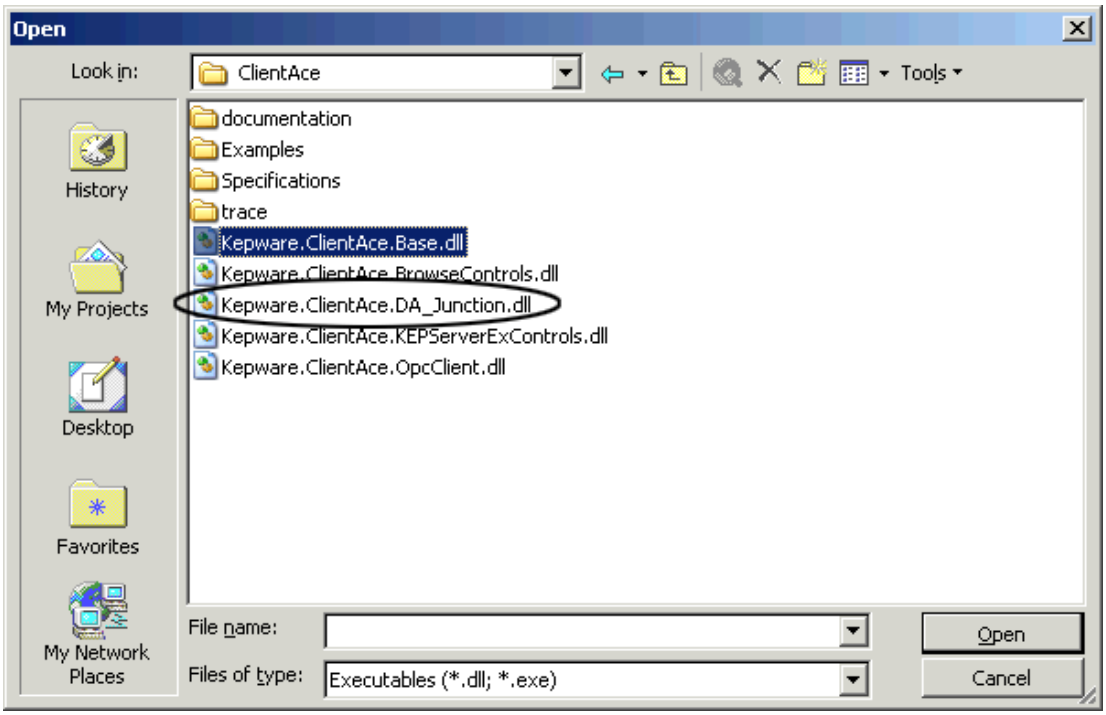


5. In the **Customize Toolbox** window, click the **Browse** button. Then, navigate to the directory where the "ClientAce.dll" files are stored.



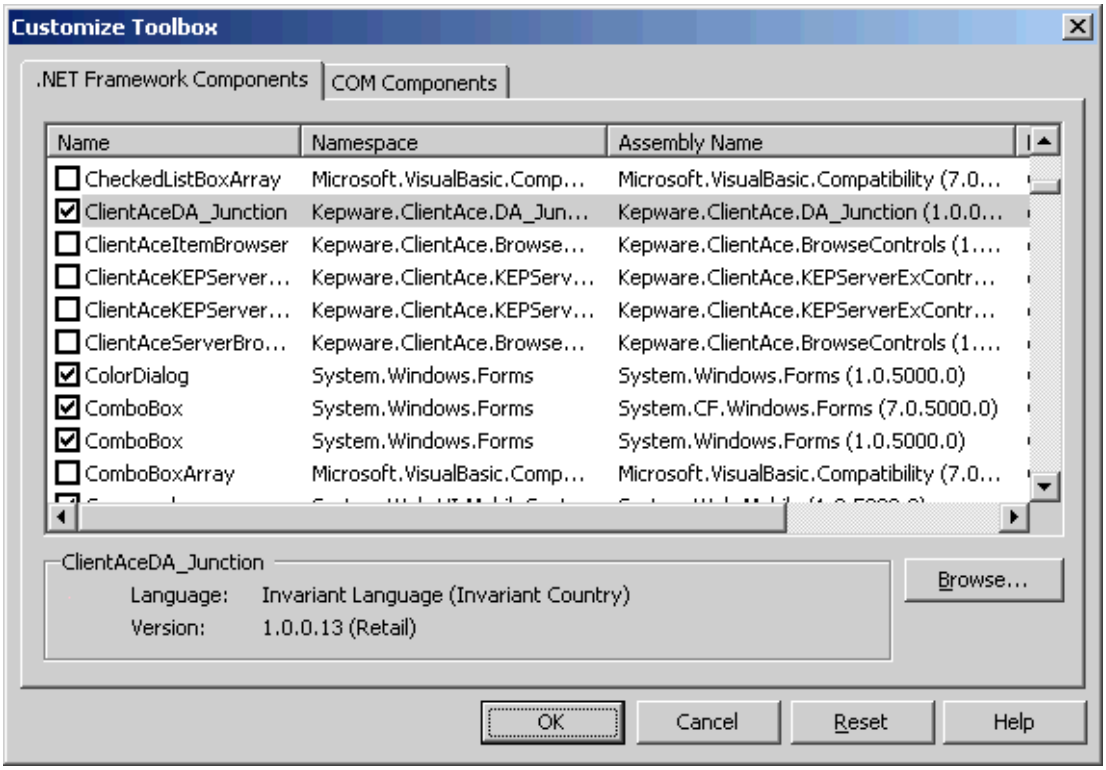
6. Click to select the .dll file that contains the controls yet to be added. Then, click **Open** (or double-click the .dll file).

   Kepware.ClientAce.DA_Junction.dll: DA Junction control
   Kepware.ClientAce.BrowseControls.dll: ServerBrowser and ItemBrowser controls
   Kepware.ClientAce.KEPServerExControls.dll: ChannelSetting and ServerState



Note: For more information, refer to **Additional Controls**.

7. Select a .dll file to display the **Customize Toolbox** window. In this example, the ClientACE.DA_Junction library is checked for inclusion.



8. To add other controls, click **Browse** and then select another .dll file. Repeat until all the control files (that is, all the .dll files) have been added to the **Customize Toolbox** for inclusion.

9. Once finished, click **OK**.

   Note: The Toolbox will display all controls that have been added.

Note: To display the applicable references in the Solution Explorer, select **View** | **Solution Explorer**. Controls that have been added to the Visual Studio Environment can also be added to the Visual Studio project by dragging them from the **Toolbox** | **ClientAce** tab onto the form. For more information, refer to **Additional Controls**.

Referencing Controls

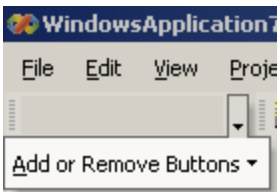
All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

Removing Blank Toolbar Options after Uninstalling ClientAce (VS 2005)

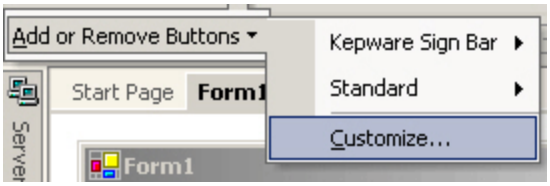
If ClientAce is uninstalled, the Microsoft Visual Studio 2005 toolbar will have a blank space where the **Sign** and **Unsign** icons were once located. For more information on removing the blank toolbar options, refer to the instructions below.

Note: This is only an issue with Visual Studio 2005 (not Visual Studio 2003).

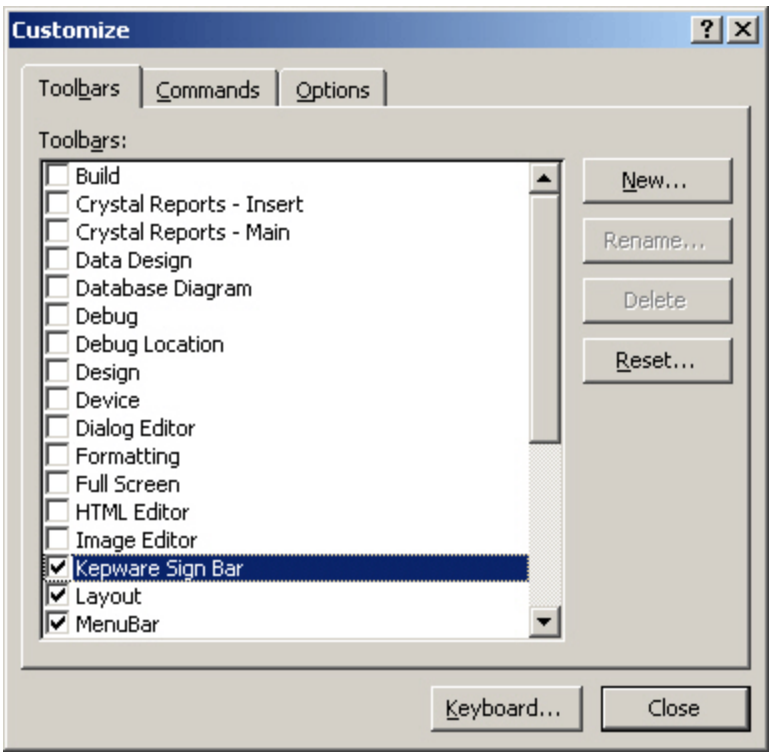
1. To start, open Visual Studio. Then, click on the small arrow on the right edge of the blank toolbar option and select **Add or Remove Buttons**.



2. Then, select **Customize**.



3. In the **Toolbars** tab, locate **Kepware Sign Bar**. Check it, and then click **Delete**.



4. Once finished, select **Close**.

Visual Studio 2008, 2010, 2012, and 2013

Creating a New Project

When creating a new project, users must set the project's Target Framework. For more information, refer to the instructions below.

1. To start, open the **Compile** tab in **My Project**.
2. Next, click **Advanced Compile Options...** | **Advanced Compiler Settings**.
    - For Visual Studio 2008, specify **.NET Framework 3.5**.
    - For Visual Studio 2010, 2012, and 2013; specify **.NET Framework 4.0**.
3. Upon completion, click **OK**.

64-Bit Operating Systems

When running a 64-bit operating system, users must set the project's Target CPU to x86. For more information, refer to the instructions below.

1. To start, open the **Compile** tab in **My Project**.
2. Then, click **Advanced Compile Options...** | **Advanced Compiler Settings** and specify **x86**.
3. Upon completion, click **OK**.

Appendix

For more information, select a link from the list below.

Deconstructing the OPC Quality Field
UAC Self Elevation

Deconstructing the OPC Quality Field

The full quality code is 16 bits: *VVVVVVVQQSSSSLL*, where:

- V is for Vendor.
- Q is for Quality.
- S is for substatus.
- L is for Limit.

Quality

QQ	Bit Value	Definition	Notes
0	00SSSSLL	Bad	The value is not useful for the reasons indicated by the substatus.
1	01SSSSLL	Uncertain	The quality of the value is uncertain for the reasons indicated by the substatus.
2	10SSSSLL	N/A	This is not used by OPC.
3	11SSSSLL	Good	The quality of the value is Good.

Note: Servers that do not support quality information must return 3 (Good). It is also acceptable for a server to return Bad or Good (0x00 or 0xC0) and to always return 0 for substatus and limit.

Substatus for Bad Quality

SSSS	Bit Value	Definition	Notes
0	000000LL	Nonspecific	The value is bad but the specific reason is unknown.
1	000001LL	Configuration Error	There is a server-specific problem with the configuration (such as, the item has been deleted from the configuration).
2	000010LL	Not Connected	The input that is required to be logically connected is missing. This quality may indicate that no value is available at this time for a reason such as the data source did not provide the value.
3	000011LL	Device Failure	A device failure has been detected.
4	000100LL	Sensor Failure	A sensor failure has been detected. The limit field may provide additional diagnostic information.
5	000101LL	Last Known Value	Communications have failed; however, the last known value is available. The age of the value can be determined from the TIMESTAMP value in OPCITEMSTATE.
6	000110LL	Communications Failure	Communications have failed. There is no last known value available.
7	000111LL	Out of Service	The block is off-scan or otherwise locked. This quality is also used when the active state of the item or the group containing the item is InActive.
8	N/A	N/A	This is not used by OPC.

Note: Servers that do not support substatus information should return 0.

Substatus for Uncertain Quality

SSSS	Bit Value	Definition	Notes
0	010000LL	Nonspecific	Indicates that there is no specific reason why the value is uncertain.
1	010001LL	Last Usable Value	Whatever was writing this value has stopped. The returned value should be regarded as "stale." Last Usable Value is different from a bad value with substatus 5 (Last Known Value), which specifically indicates a detectable communications error on a "fetched" value. Last Usable Value indicates the failure of some external source to

			send a value within an acceptable period of time. The age of the value can be determined from the TIMESTAMP value in OPCITEMSTATE.
2-3	N/A	N/A	This is not used by OPC.
4	010100LL	Sensor Not Accurate	Either the value has "pegged" at one of the sensor limits (in which case the limit field should be set to 1 or 2) or the sensor is otherwise known to be out of calibration as indicated by some form of internal diagnostics (in which case the limit field should be 0).
5	010101LL	Engineering Units Exceeded	The value returned is outside of the limits defined for that parameter. In this case, the limit field indicates which limit has been exceeded; however, that does not necessarily mean that the value cannot move farther out of range.
6	010110LL	Sub-normal	The value is derived from multiple sources and has less than the required number of good sources.
7-15	N/A	N/A	This is not used by OPC.

Note: Servers that do not support substatus information should return 0.

Substatus for Good Quality

SSSS	Bit Value	Definition	Notes
0	110000LL	Nonspecific	The value is good and there are no special conditions.
1-5	N/A	N/A	This is not used by OPC.
6	110110LL	Local Override	The value has been overridden. This is generally because the input has been disconnected and a manually entered value has been "forced."
7-15	N/A	N/A	This is not used by OPC.

Note: Servers that do not support substatus information should return 0.

Limit

LL	Bit Value	Definition	Notes
0	QQSSSS00	Not Limited	The value is free to move up or down.
1	QQSSSS01	Low Limited	The value has "pegged" at some lower limit.
2	QQSSSS10	High Limited	The value has "pegged" at some high limit.
3	QQSSSS11	Constant	The value is constant and cannot move.

Note: The limit value is valid regardless of the quality and substatus values. In some cases (such as Sensor Failure), the limit value can provide useful diagnostic information. Servers that do not support limit information should return 0.

UAC Self Elevation

When developing applications to run on Windows Vista and higher operating systems with UAC enabled, there will be many times that the application needs to execute functions and processes that require Administrator privileges. By default, .NET applications are configured to run with the privileges of the invoker. When UAC is enabled, this is referred to as a user. To allow the application to self-elevate, the developer must create an Application Manifest.

Creating the Application Manifest in Visual Basic .NET (VB.NET)

In Visual Basic .NET, the Application Manifest is generated through the user interface.

1. In the **Solution Explorer**, right-click on the project and then select **Properties**. Then, open the **Application** tab.
2. Next, click **View Windows Settings**.

   Note: The Application Manifest will be generated and opened in the Project View.

Creating an Application Manifest in C-Sharp (C#)

In C-Sharp, the Application Manifest is generated through the menu.

1. In the **Solution Explorer**, right-click on the project and then select **Add**.
2. Next, select **New Item**.
3. In **Add New Item**, select **Application Manifest file**. Then, click **Add**.

   Note: The Application Manifest will be generated and opened in the Project View.

Editing the Application Manifest

The Application Manifest is an XML-formatted file. When it is generated, the "requestedExecutionLevel" is set to "asInvoker" by default. On systems where the invoker is an Administrator, the application can usually run after changing the setting to "highestAvailable". In some cases, other aspects of the project may require the setting "requireAdministrator". It is recommended that users edit the Application Manifest according to the needs of the project.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv1="urn:schemas-
microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
            If you want to change the Windows User Account Control level replace the
            requestedExecutionLevel node with one of the following.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

            Specifying requestedExecutionLevel node will disable file and registry virtualization.
            If you want to utilize File and Registry Virtualization for backward
            compatibility then delete the requestedExecutionLevel node.
        -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>

  <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
      <!-- A list of all Windows versions that this application is designed to work with. Windows will
automatically select the most compatible environment.-->

      <!-- If your application is designed to work with Windows 7, uncomment the following supportedOS
node-->
      <!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>-->

    </application>
  </compatibility>

  <!-- Enable themes for Windows common controls and dialogs (Windows XP and later) -->
  <!--  <dependency>
    <dependentAssembly>
      <assemblyIdentity
          type="win32"
          name="Microsoft.Windows.Common-Controls"
          version="6.0.0.0"
          processorArchitecture="*"
          publicKeyToken="6595b64144ccf1df"
          language="*"
        />
    </dependentAssembly>
  </dependency>-->

</asmv1:assembly>
```

Index

A

B

C

D

E

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W